Principles of Software Construction: Objects, Design, and Concurrency

Managing change (3)

Charlie Garrod **Bogdan Vasilescu**

School of Computer Science
institute for SOFTWARE RESEARCH

Administrivia

- ~~Homework 6 checkpoint deadline yesterday (Monday, April 30th)~~
- Homework 6 due Wednesday, May 2nd
- Final exam Monday May 7th 5:30-8:30 PH 100
- Review session Saturday May 5th 2pm WH 5403

Key concepts from Thursday

Aside: Which files to manage

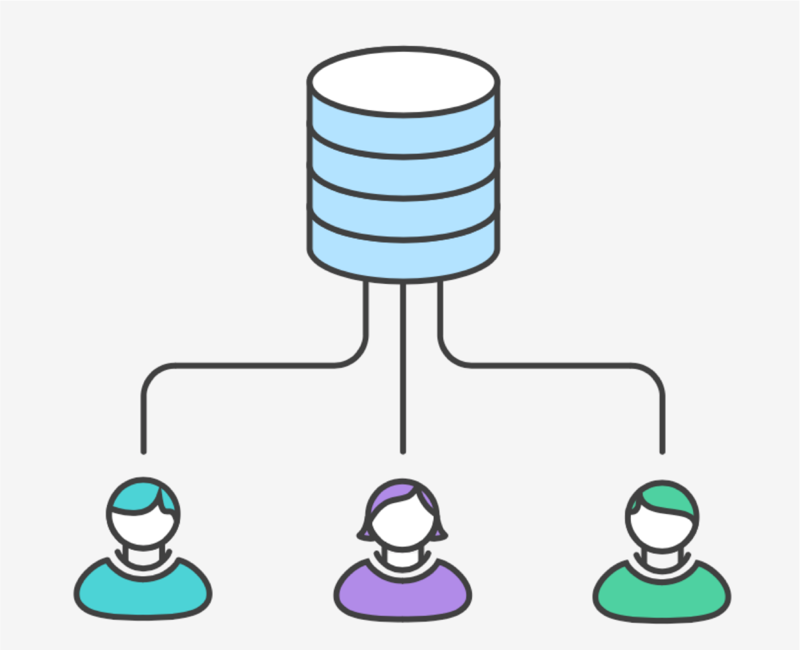
- All code and noncode files
  - Java code
  - Build scripts
  - Documentation
- Exclude generated files (.class, …)
- Most version control systems have a mechanism to exclude files (e.g., .gitignore)

BRANCH WORKFLOWS

https://www.atlassian.com/git/tutorials/comparing-workflows

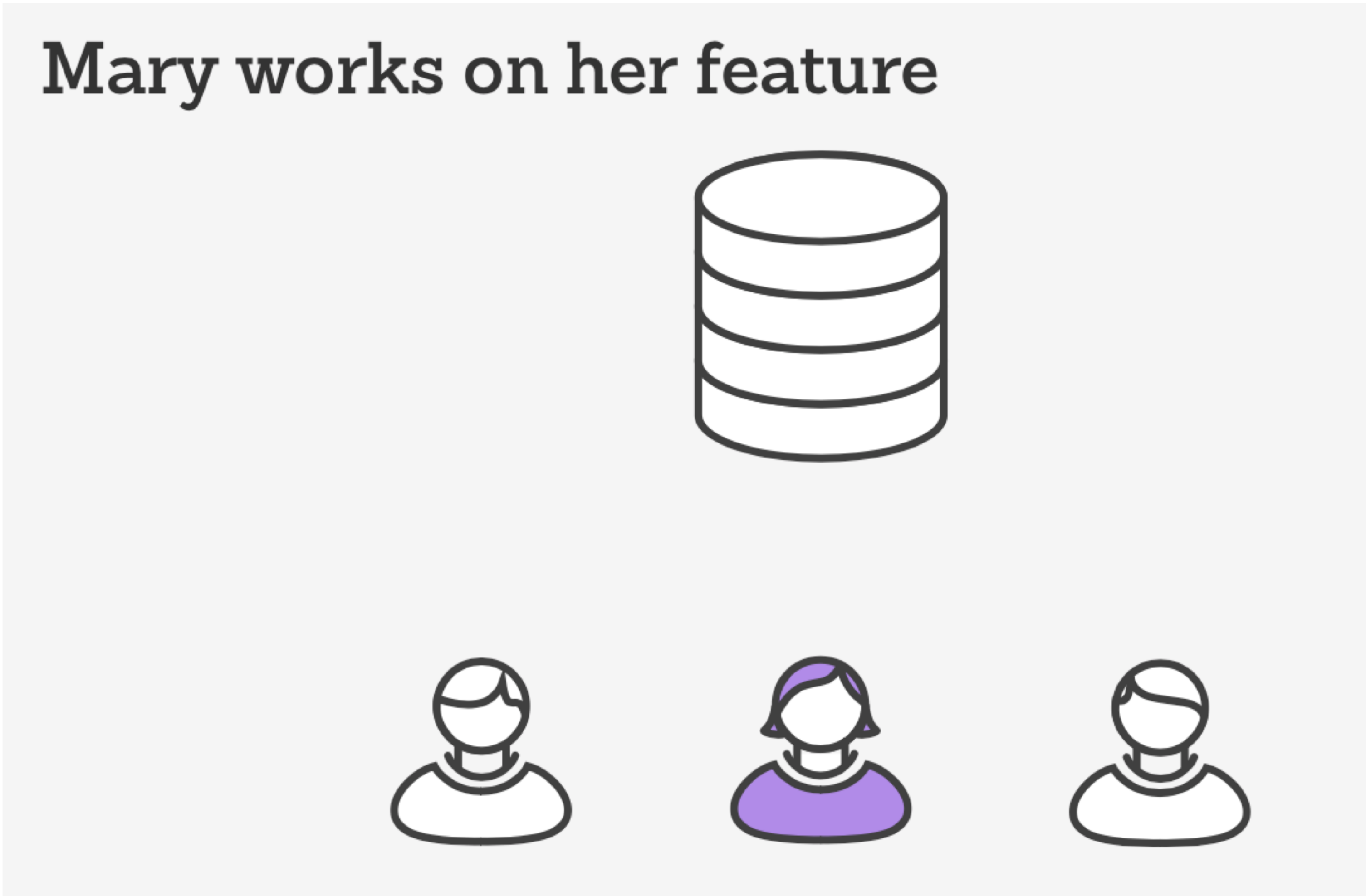
1. Centralized workflow

- Central repository to serve as the single point-of-entry for all changes to the project
- Default development branch is called master
  - all changes are committed into master
  - doesn’t require any other branches

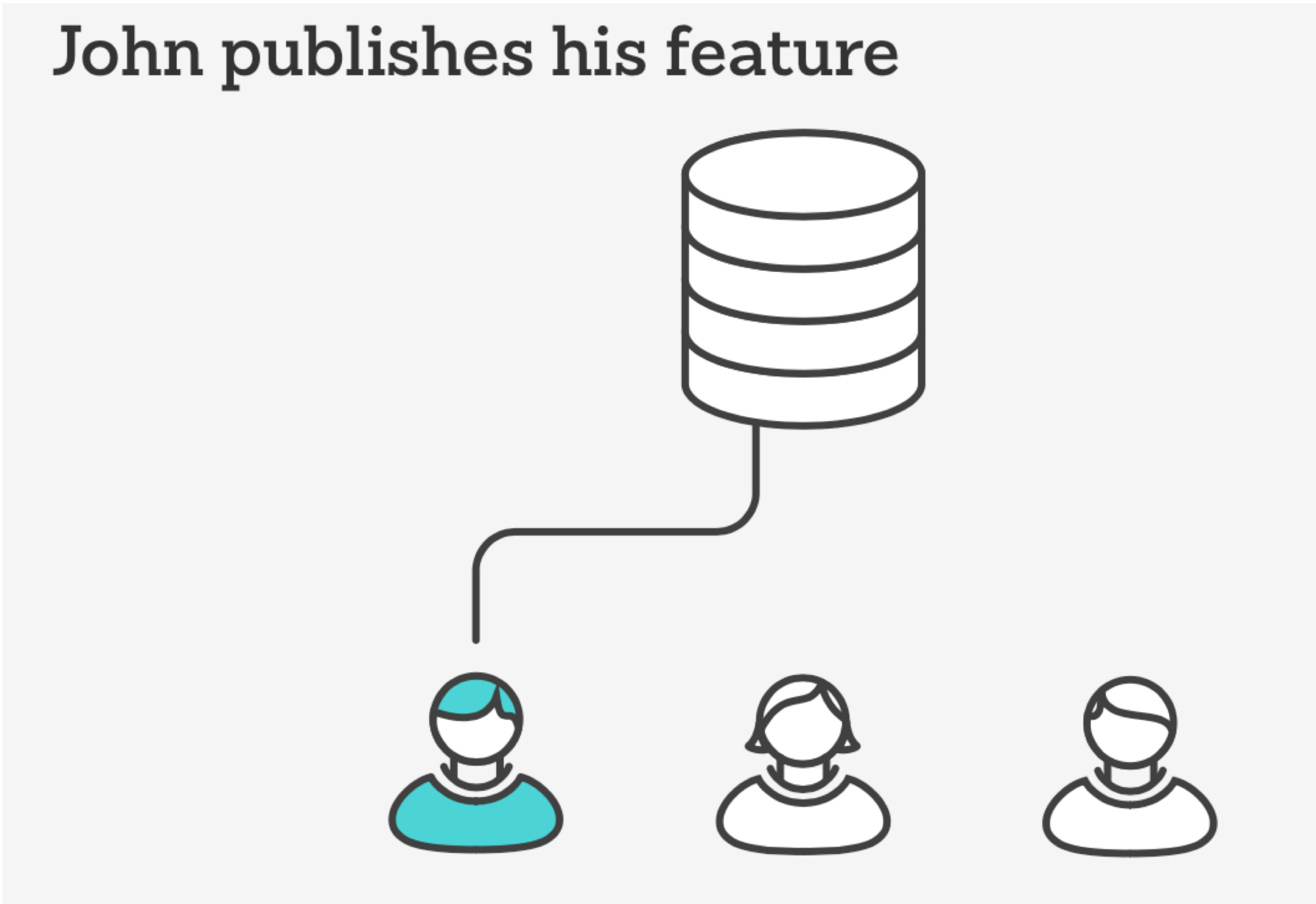


Example

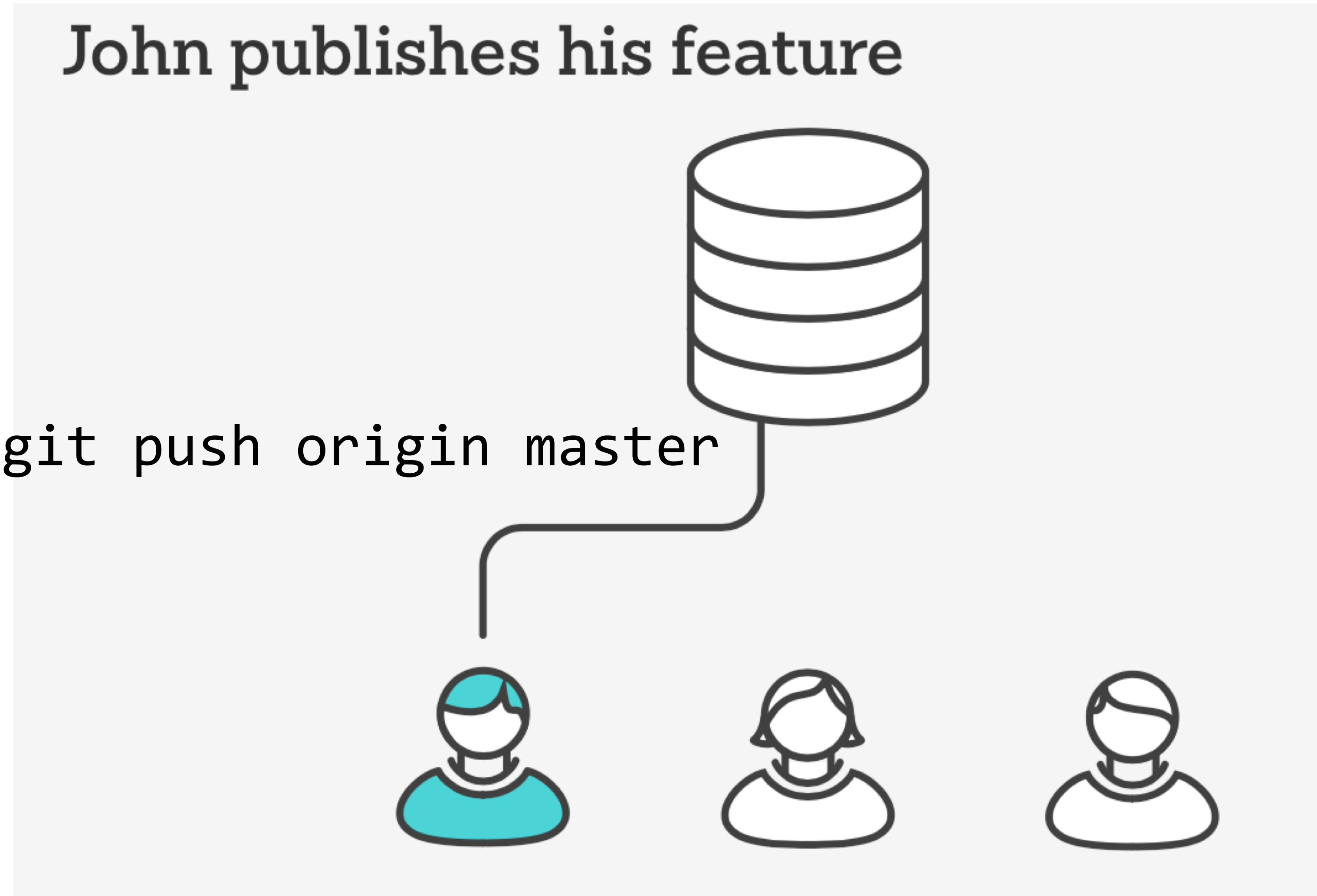
Example



Example

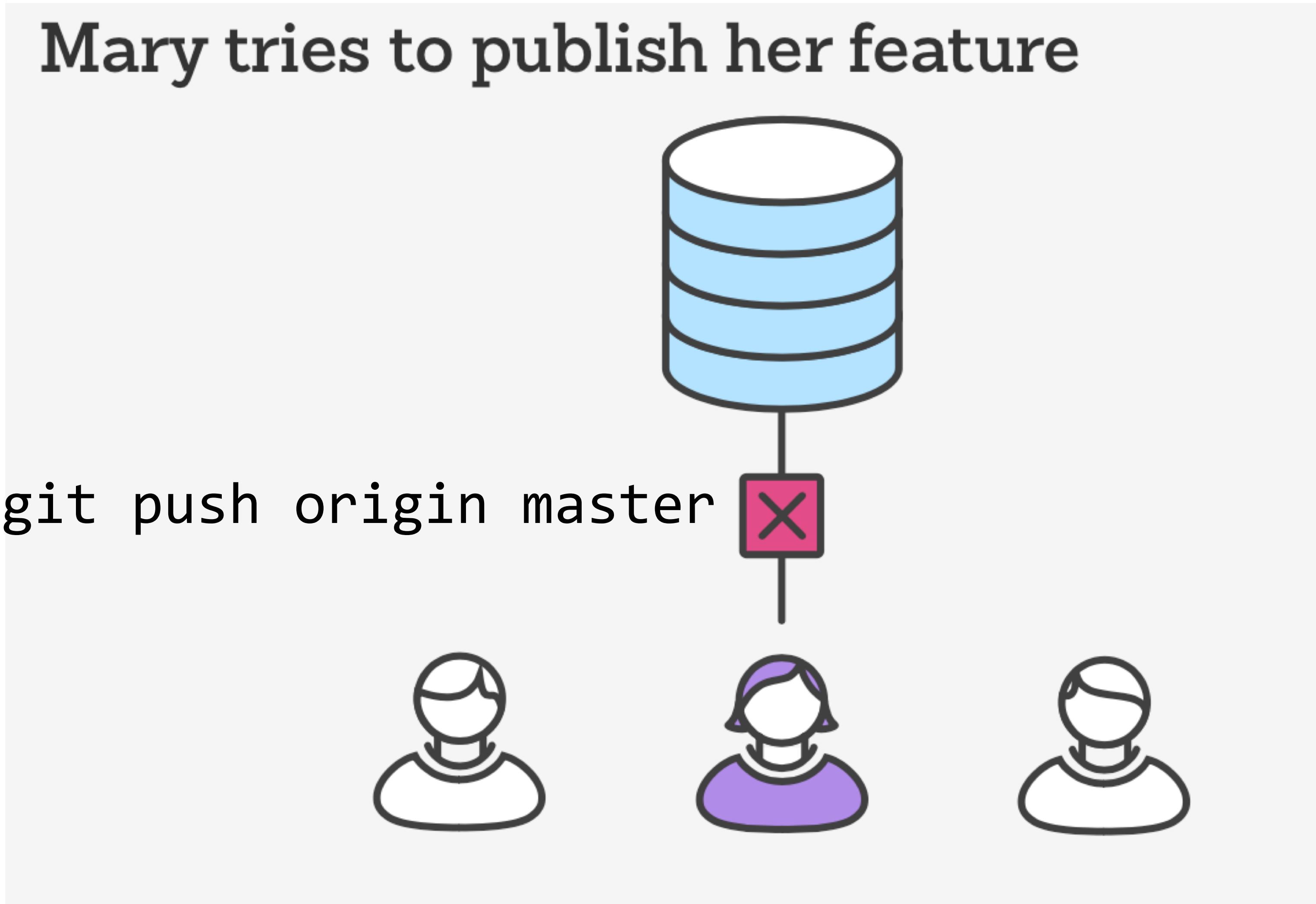


Example



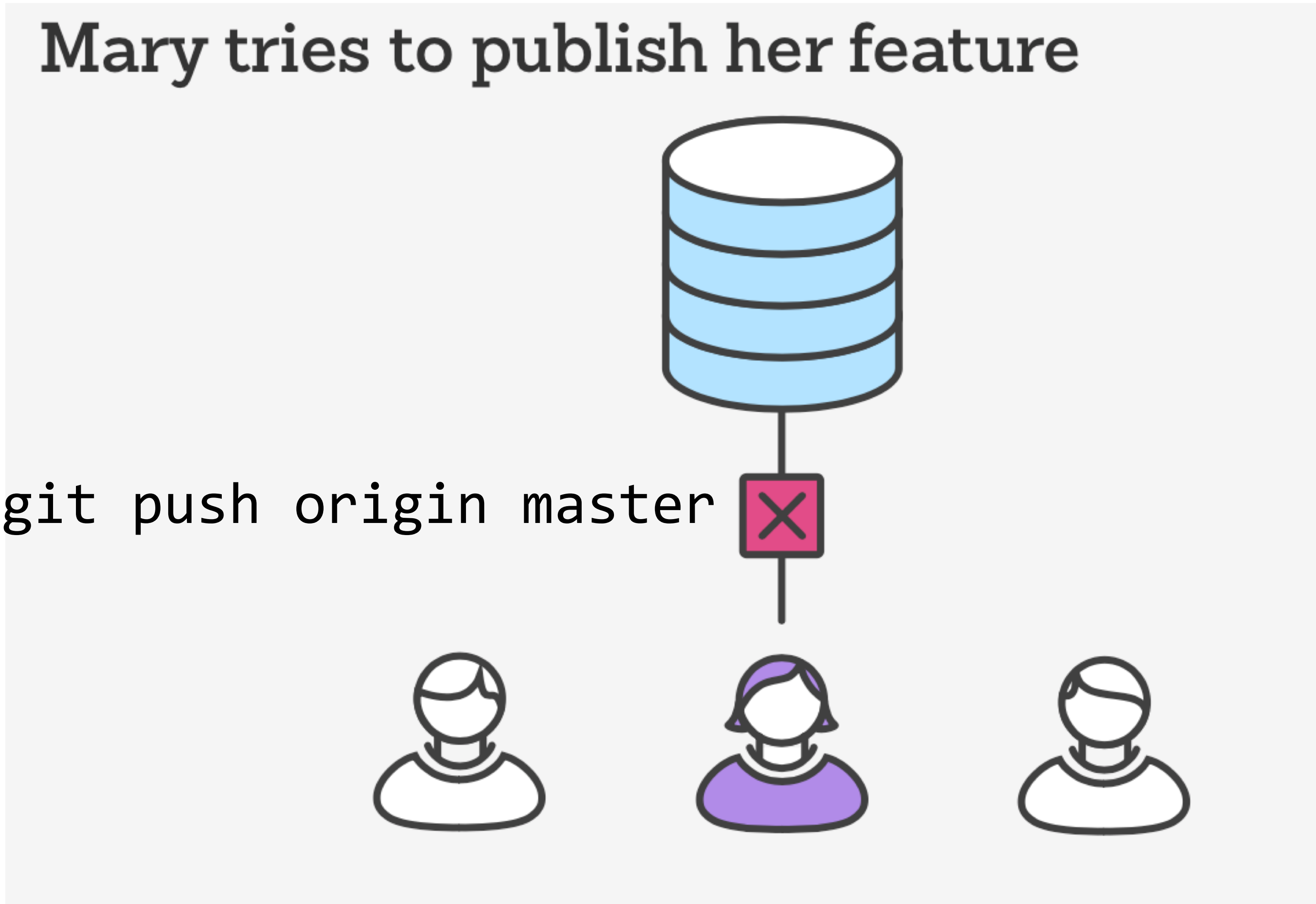

```
git push origin master
```

Example



git push origin master

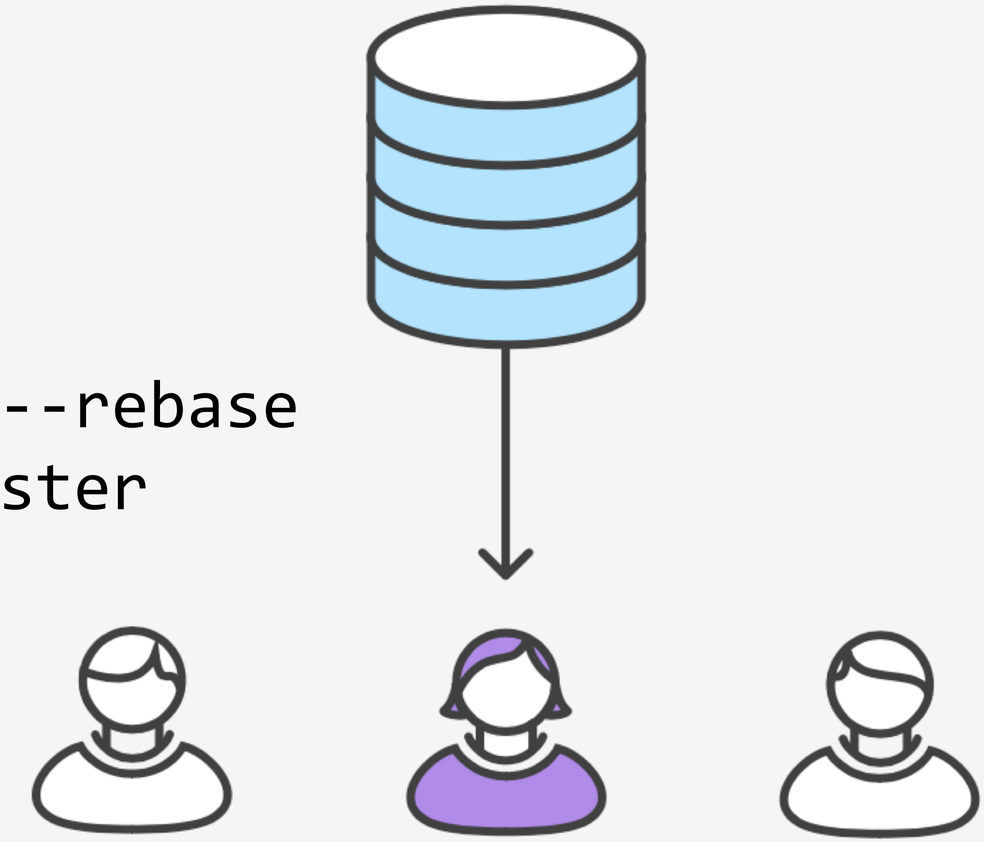
```
error: failed to push some refs to '/path/to/repo.git' hint: Updates were rejected because the tip of your current branch is behind hint: its remote counterpart. Merge the remote changes (e.g. 'git pull') hint: before pushing again. hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

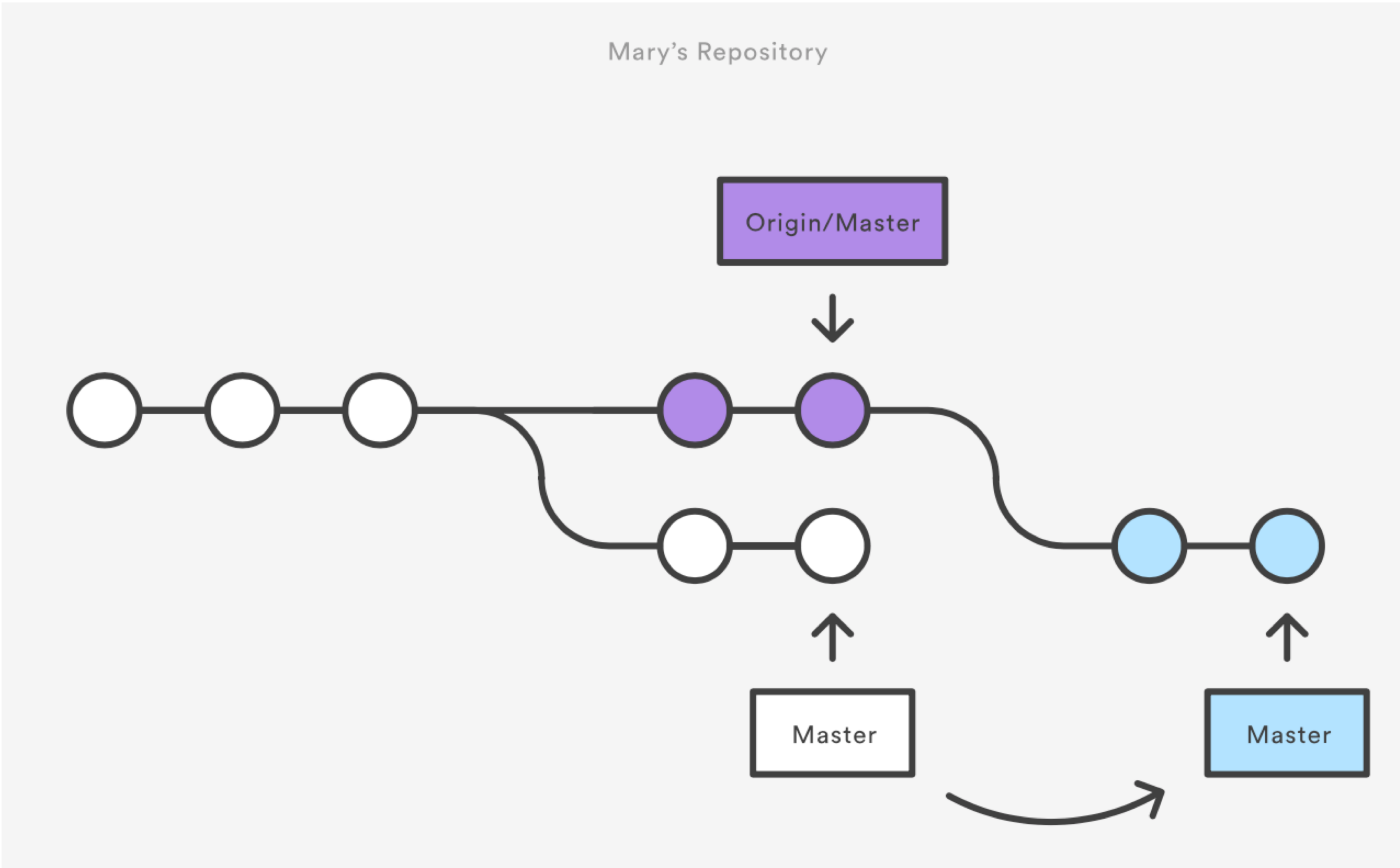


Example

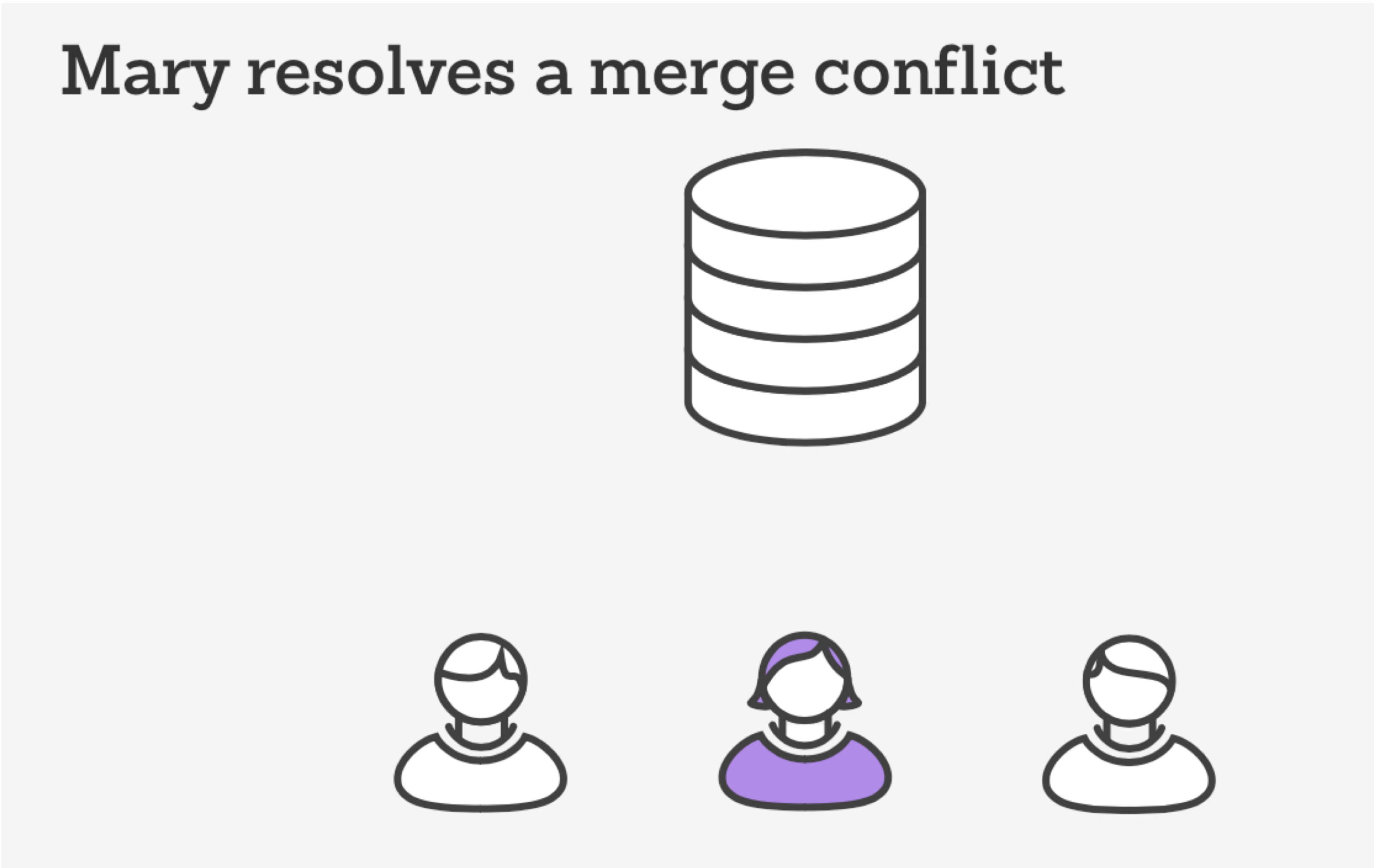
Mary rebases on top of John's commit(s)

```
git pull --rebase
origin master
```

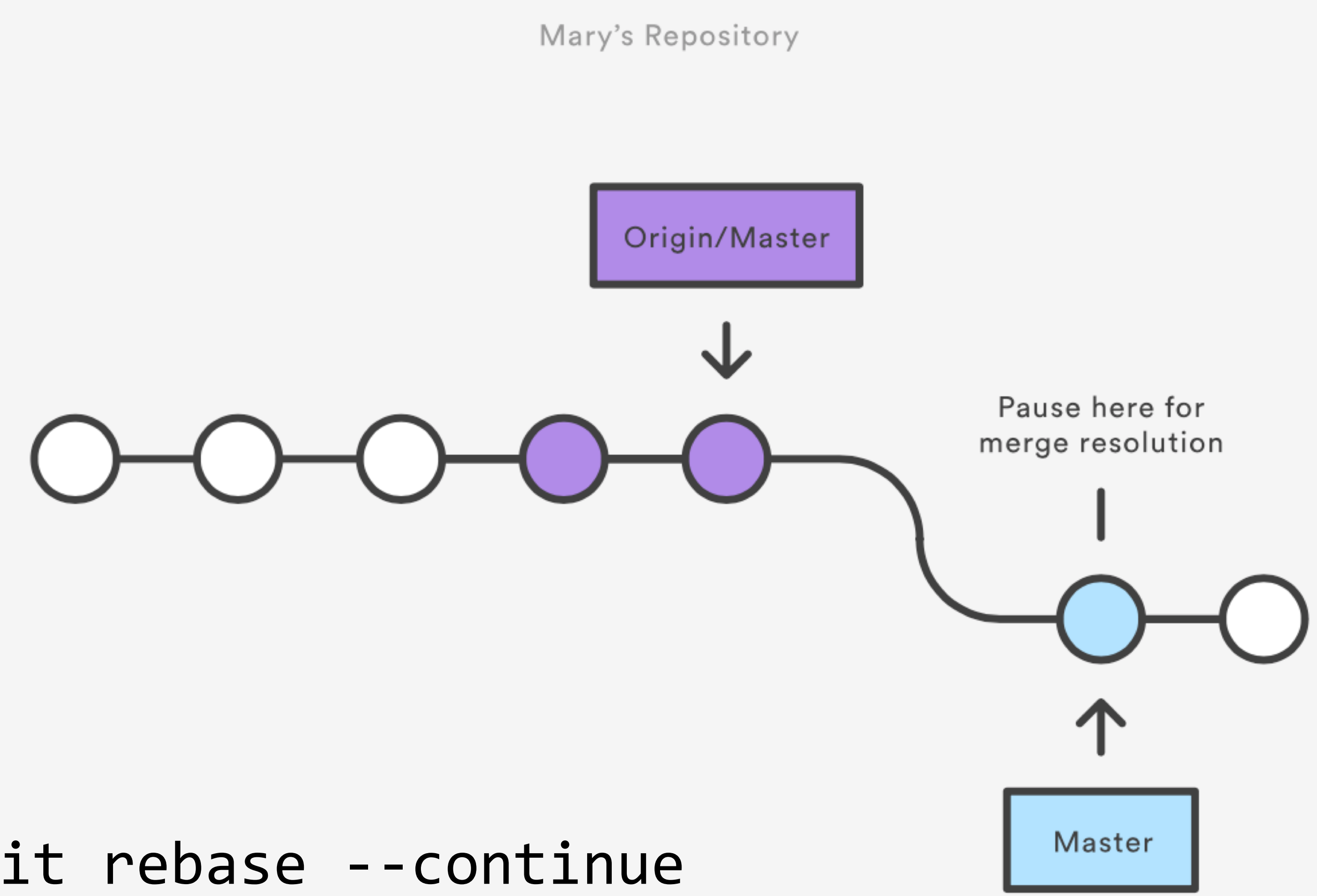



Mary's Repository
Origin/Master
Master
Master

Example

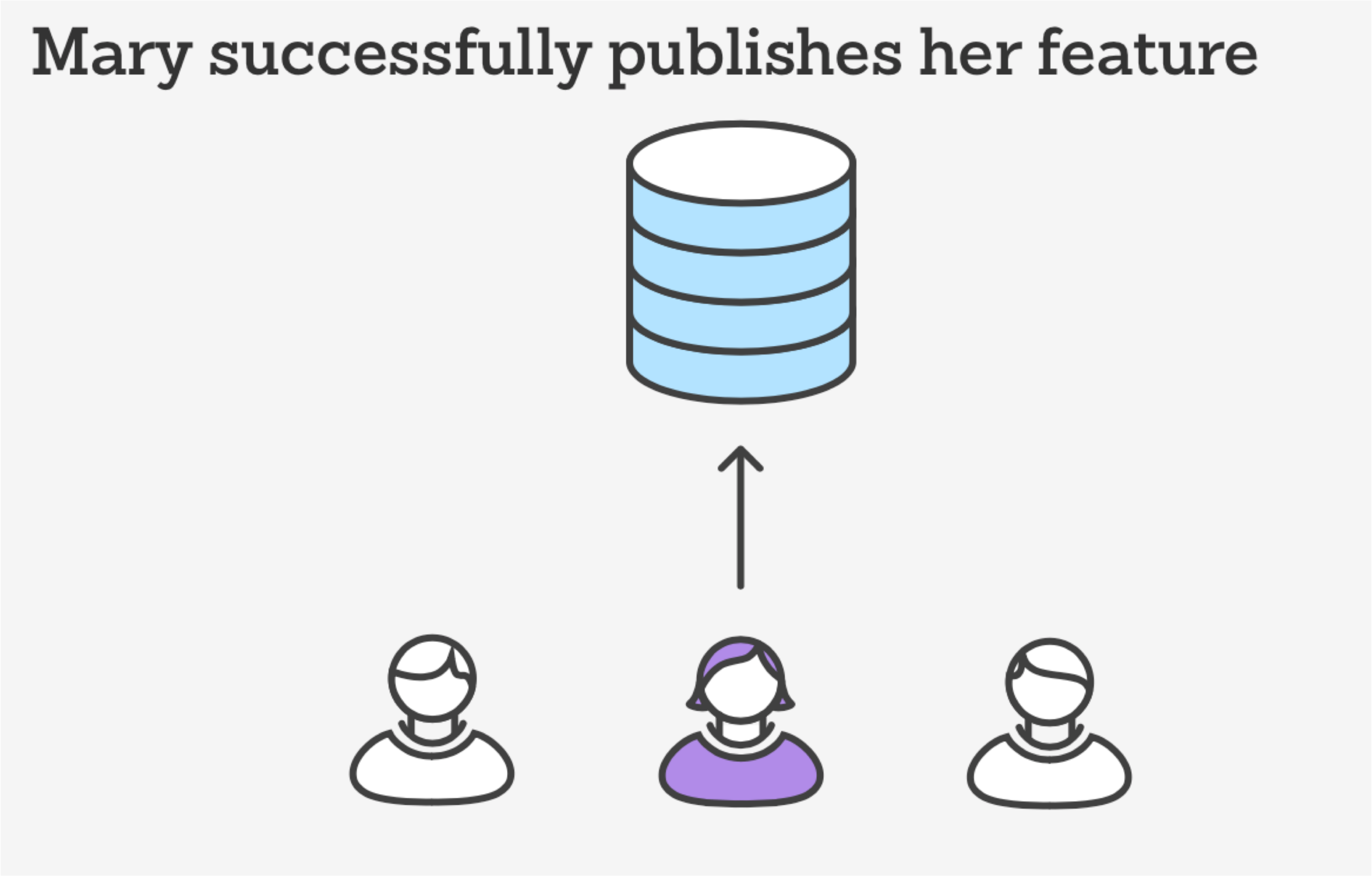


Example




```
git rebase --continue
```

Example



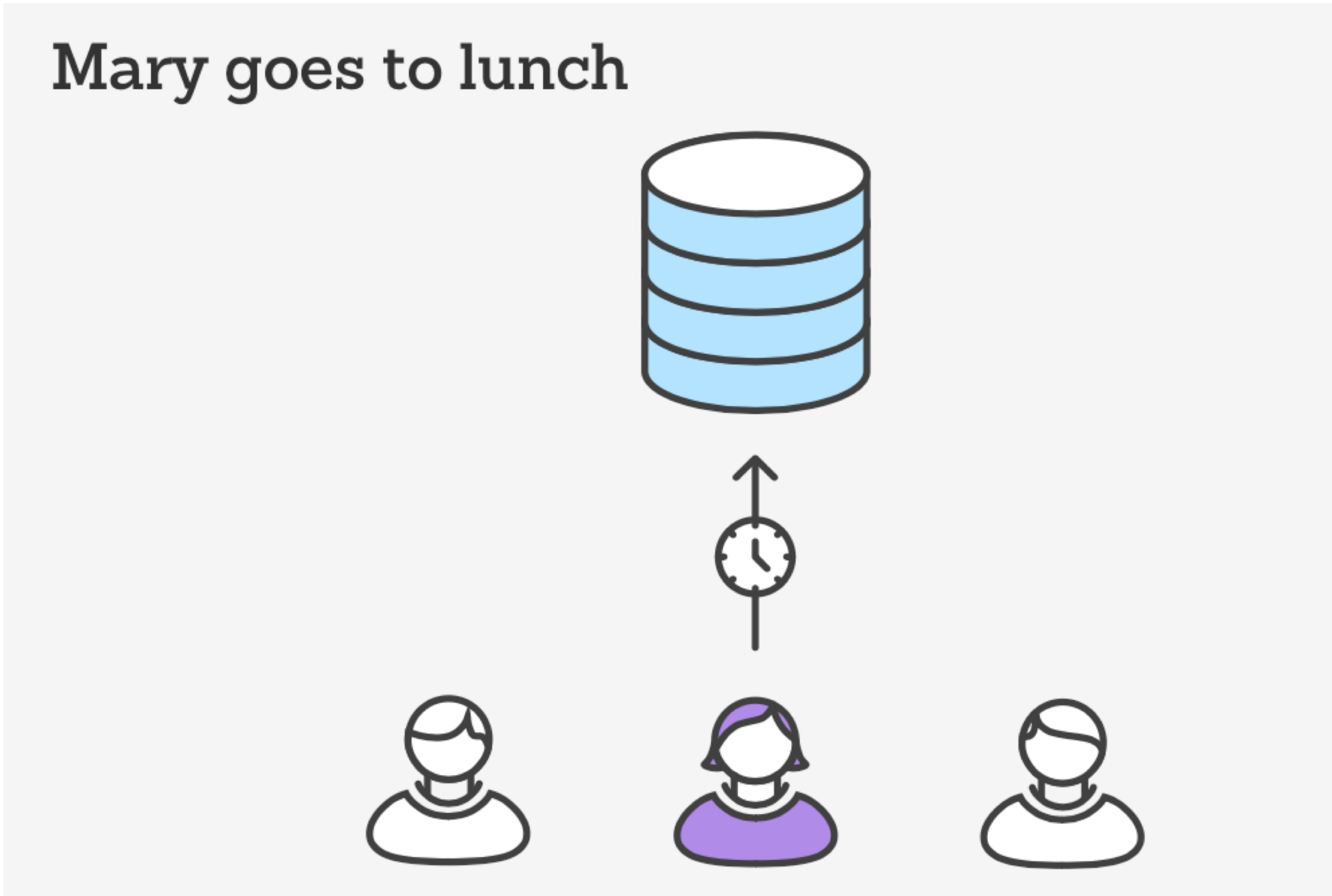
2. Git Feature Branch Workflow

- *All* feature development should take place in a dedicated branch instead of the master branch
- Multiple developers can work on a particular feature without disturbing the main codebase
  - master branch will never contain broken code (enables CI)
  - Enables pull requests (code review)

Example

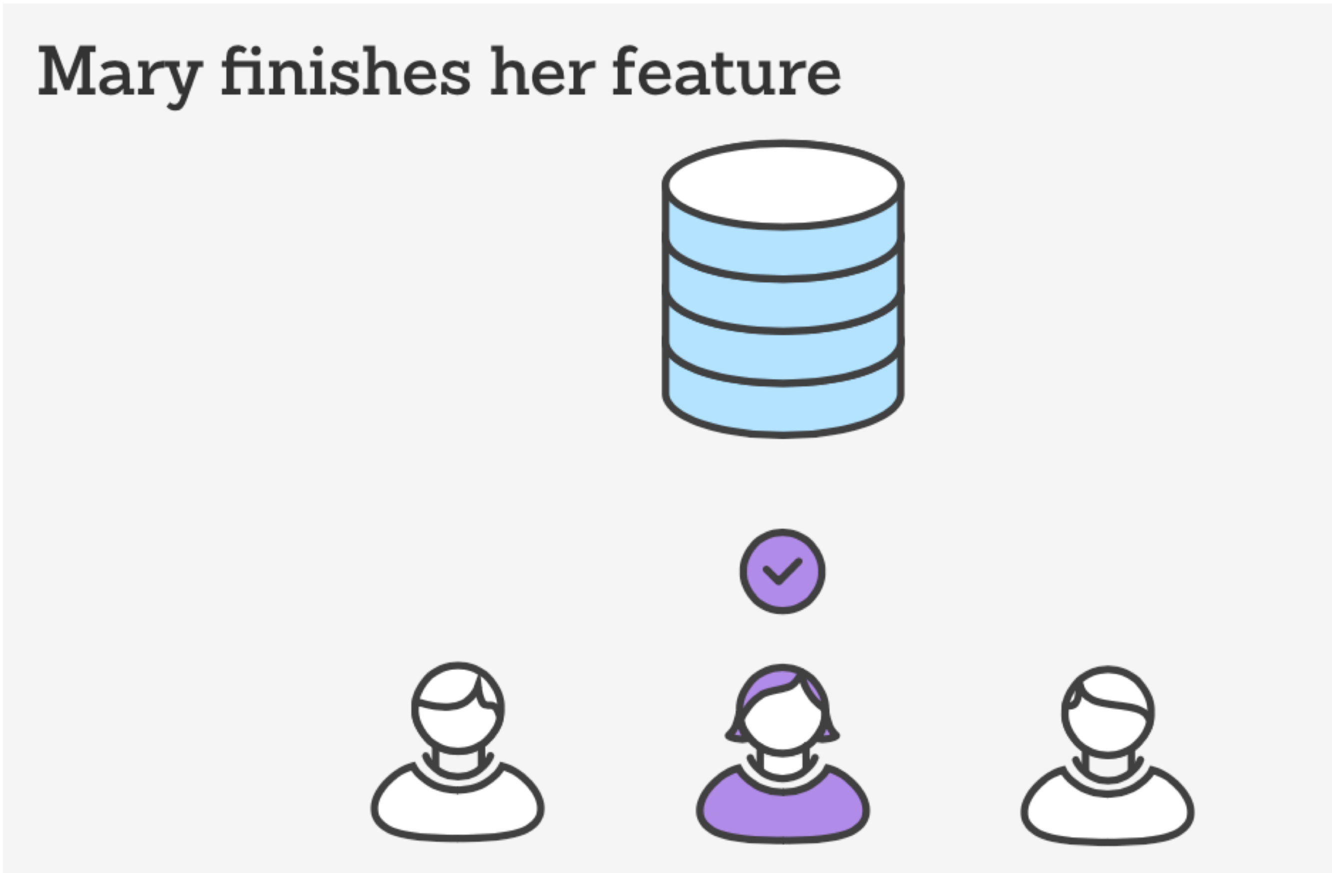

```
git checkout -b marys-feature master

git status
git add <some-file>
git commit
```

Example



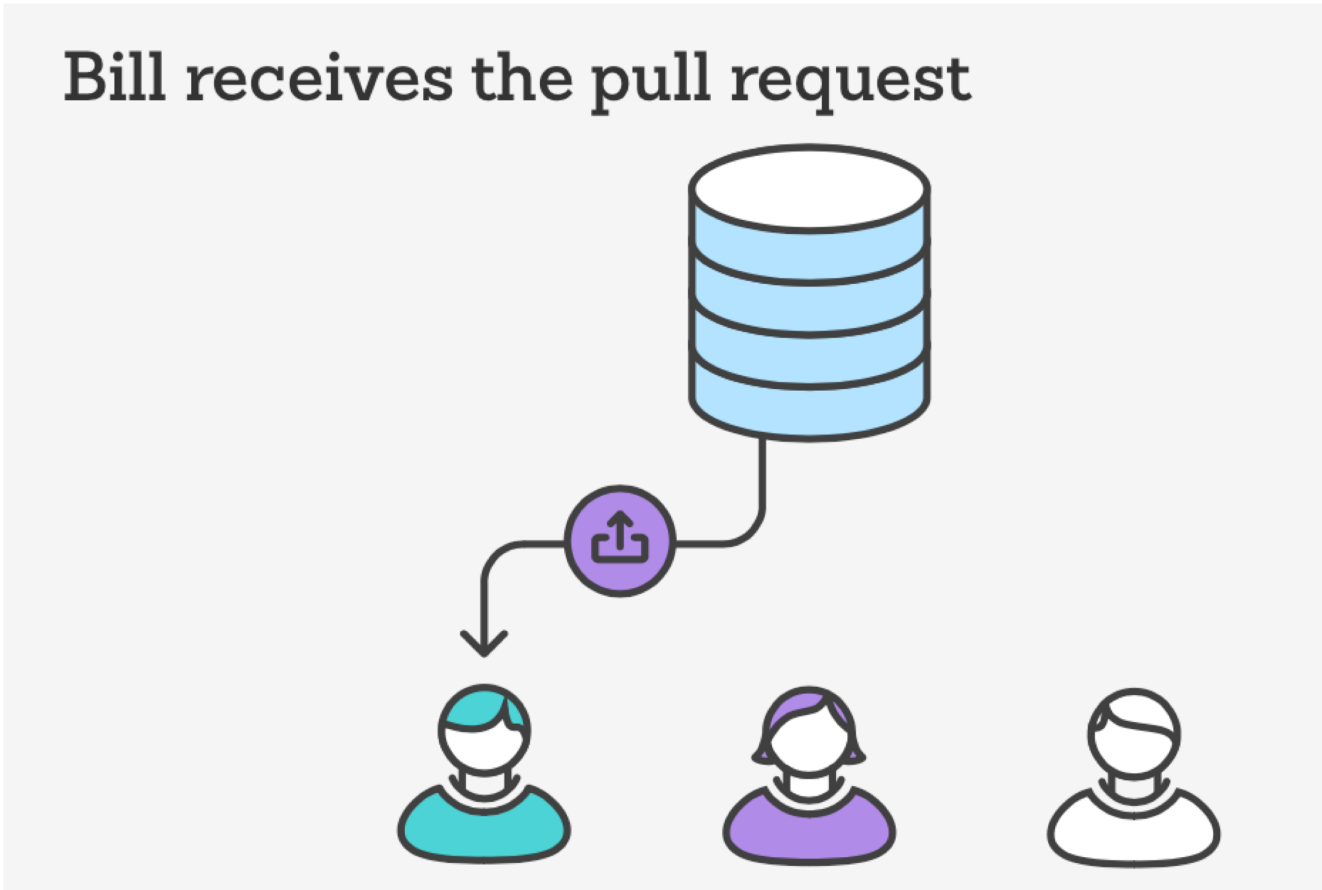

```
git push -u origin marys-feature
```

Example

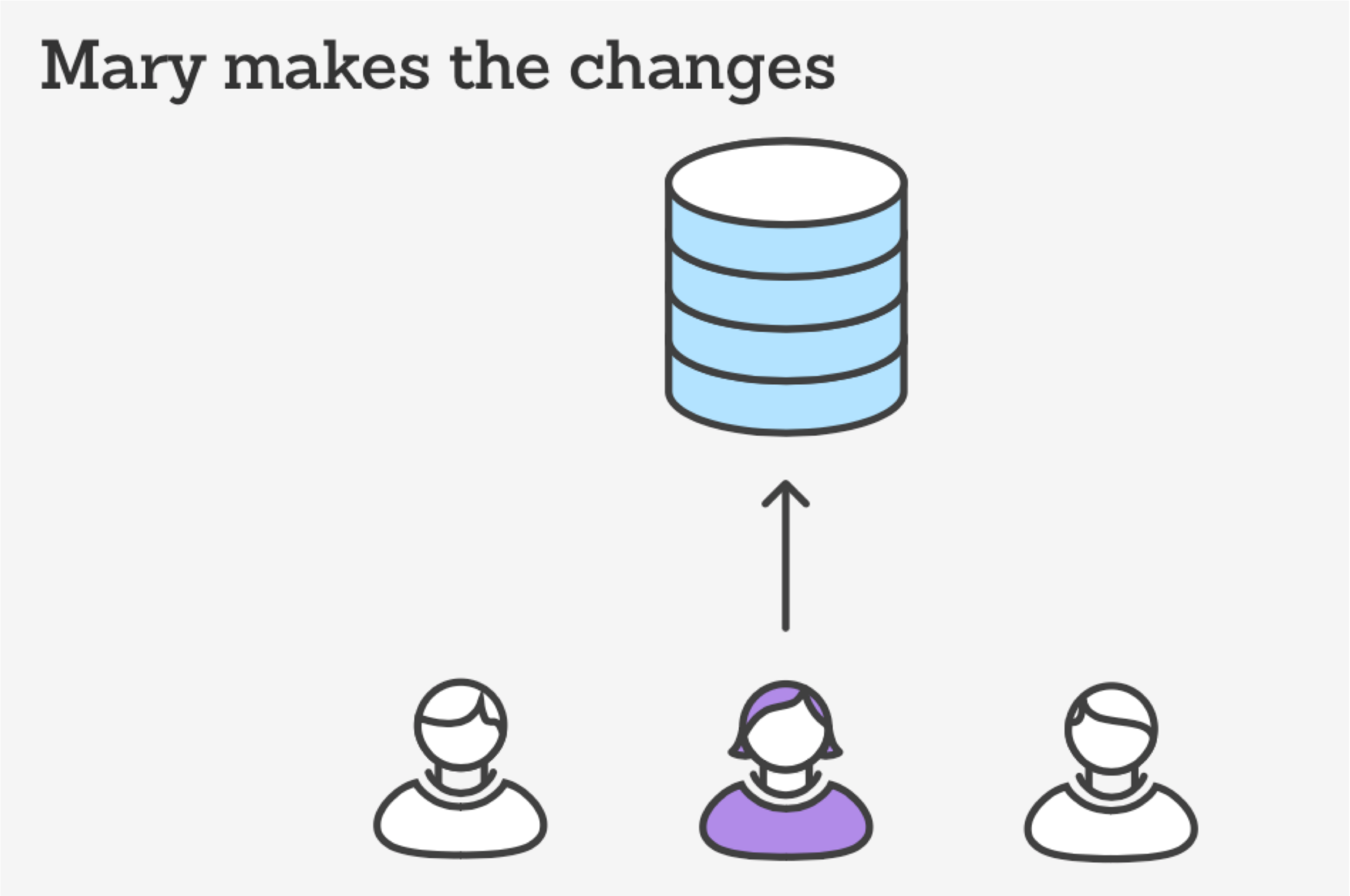


`git push`

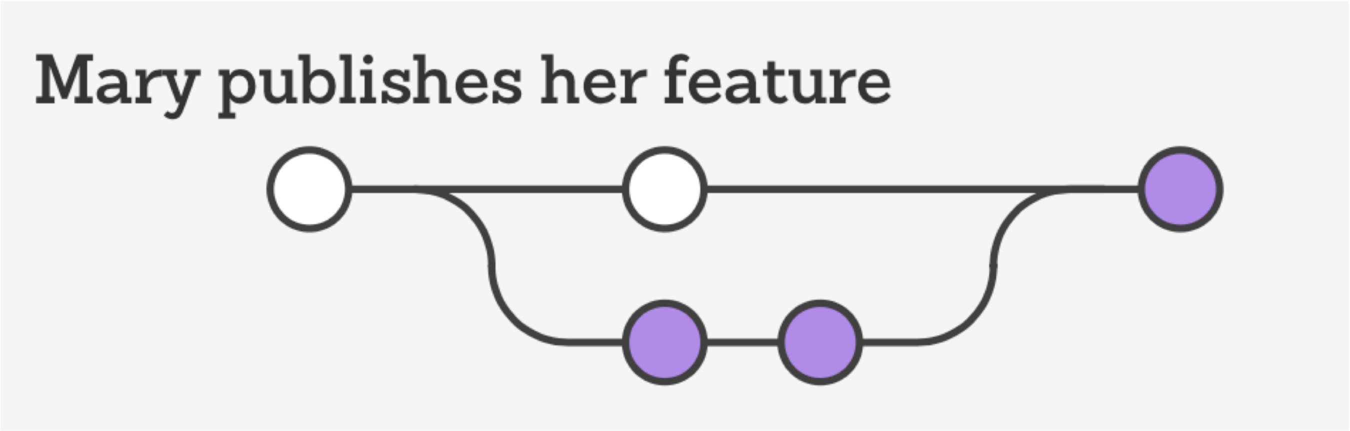
Example



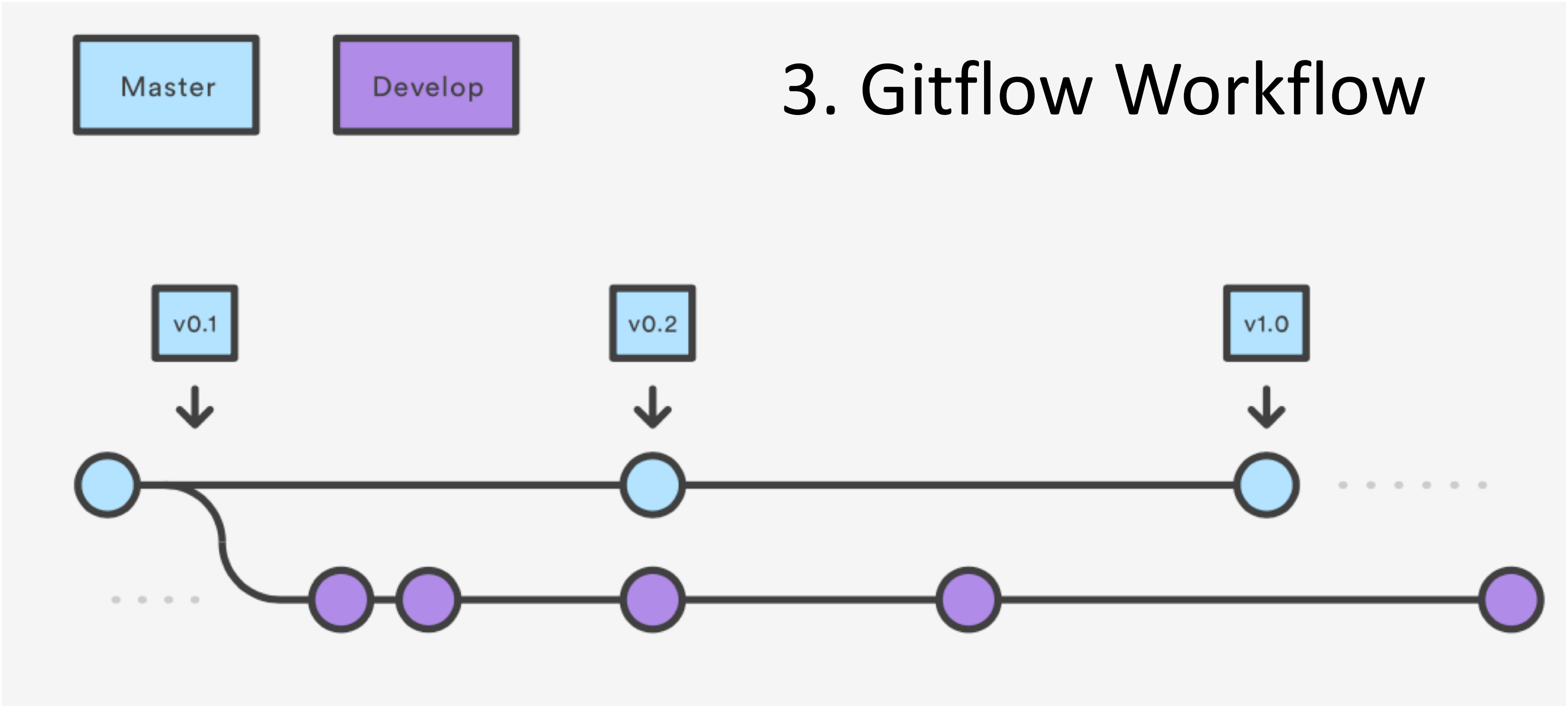
Example



Example - Merge pull request

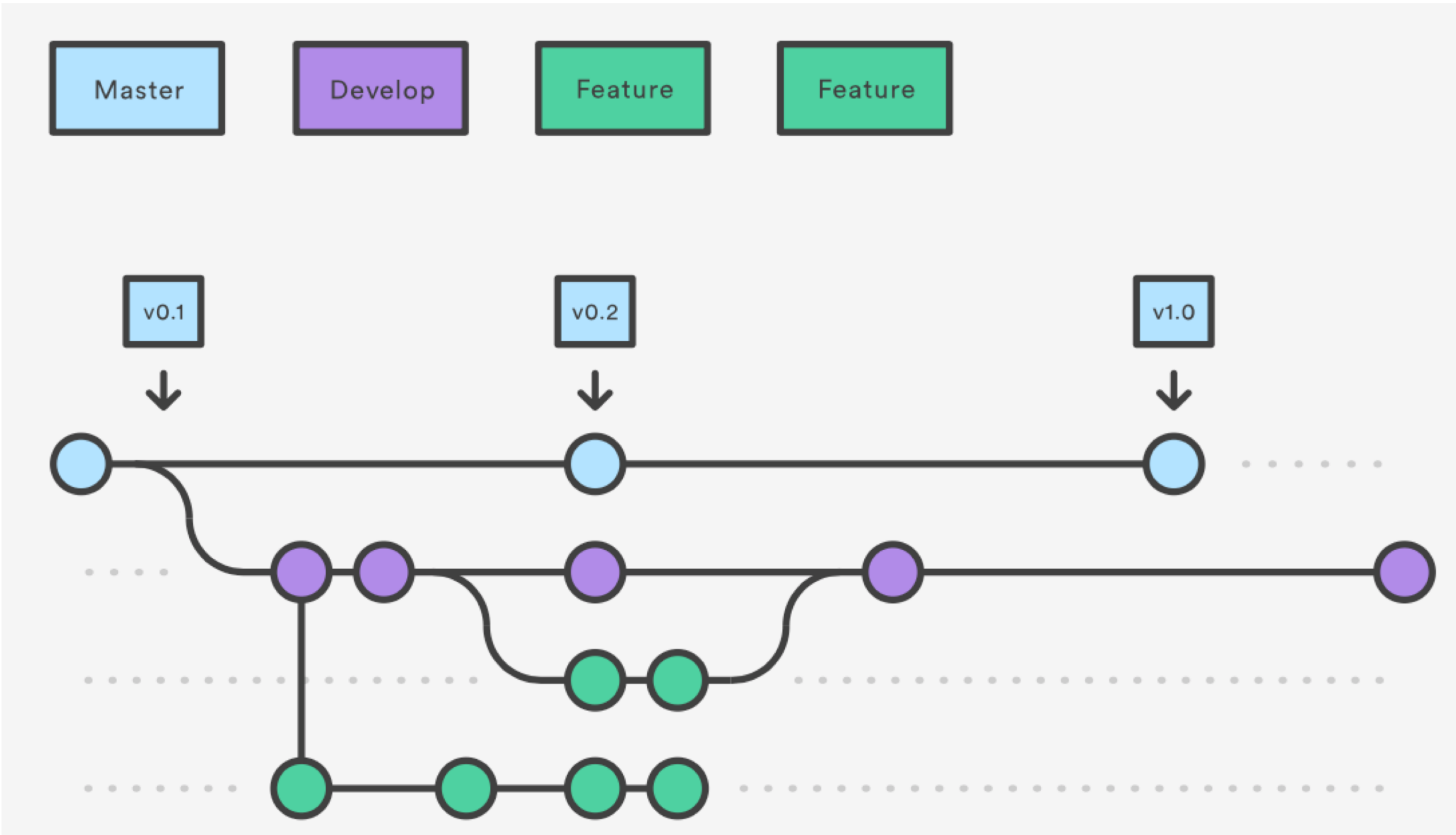



```
git checkout master
git pull
git pull origin marys-feature
git push
```

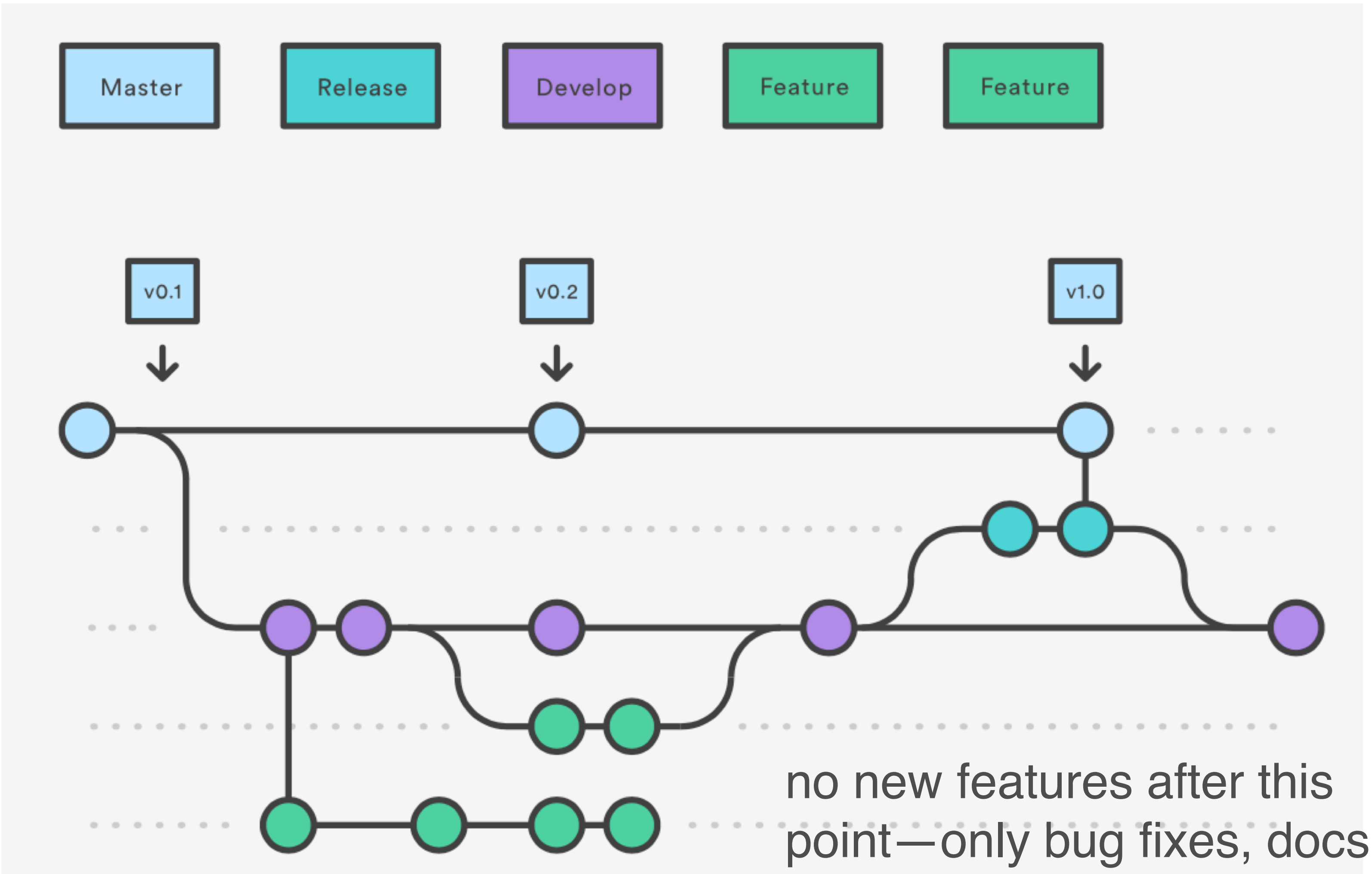


- Strict branching model designed around the project release
  - Suitable for projects that have a scheduled release cycle
- Branches have specific roles and interactions
- Uses two branches
  - master stores the official release history; tag all commits in the master branch with a version number
  - develop serves as an integration branch for features

GitFlow feature branches (from develop)



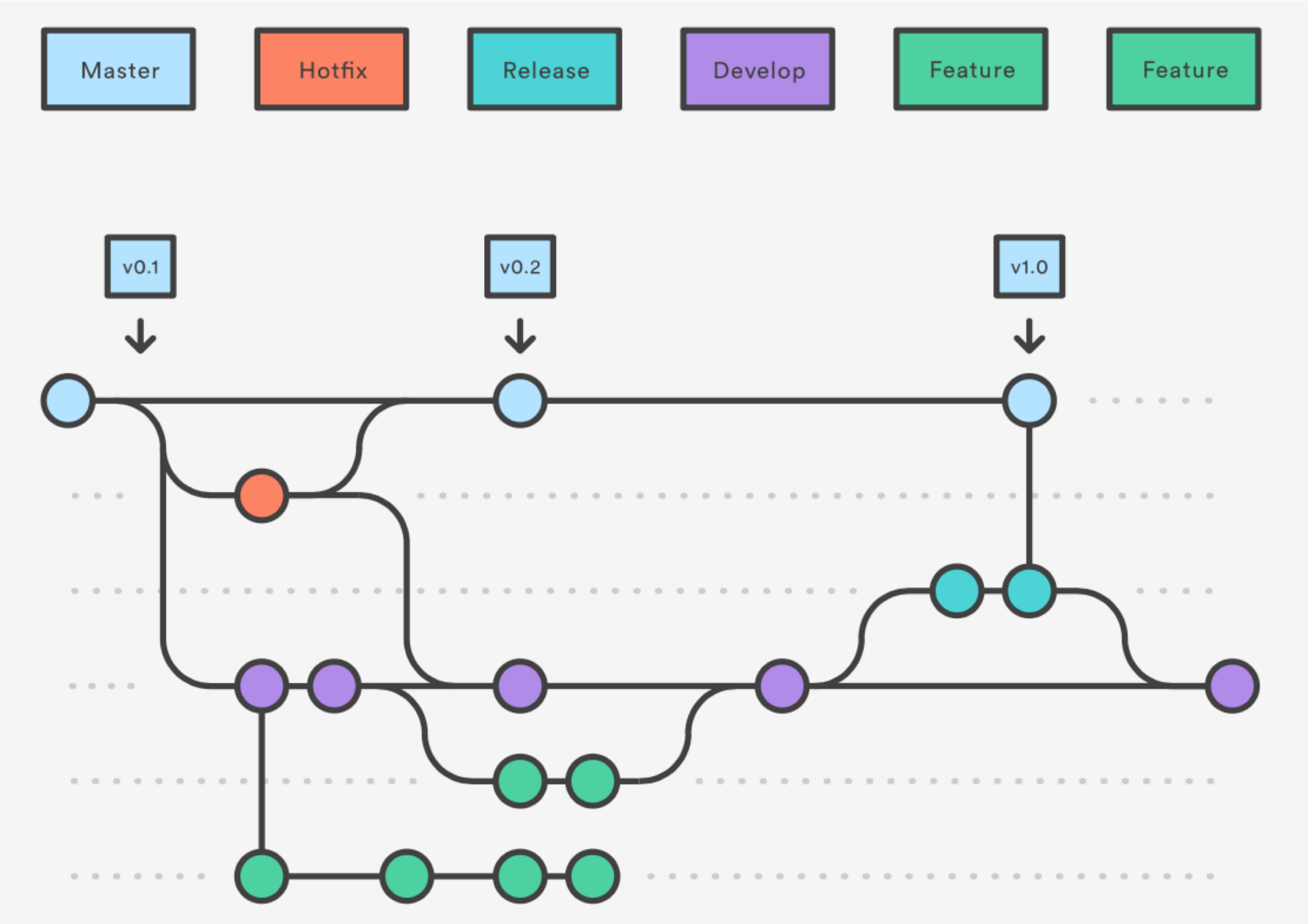
GitFlow release branches (eventually into master)



no new features after this point—only bug fixes, docs, and other release tasks

GitFlow hotfix branches

used to quickly patch production releases



Summary

- Version control has many advantages
  - History, traceability, versioning
  - Collaborative and parallel development
- Collaboration with branches
  - Different workflows
- From local to central to distributed version control

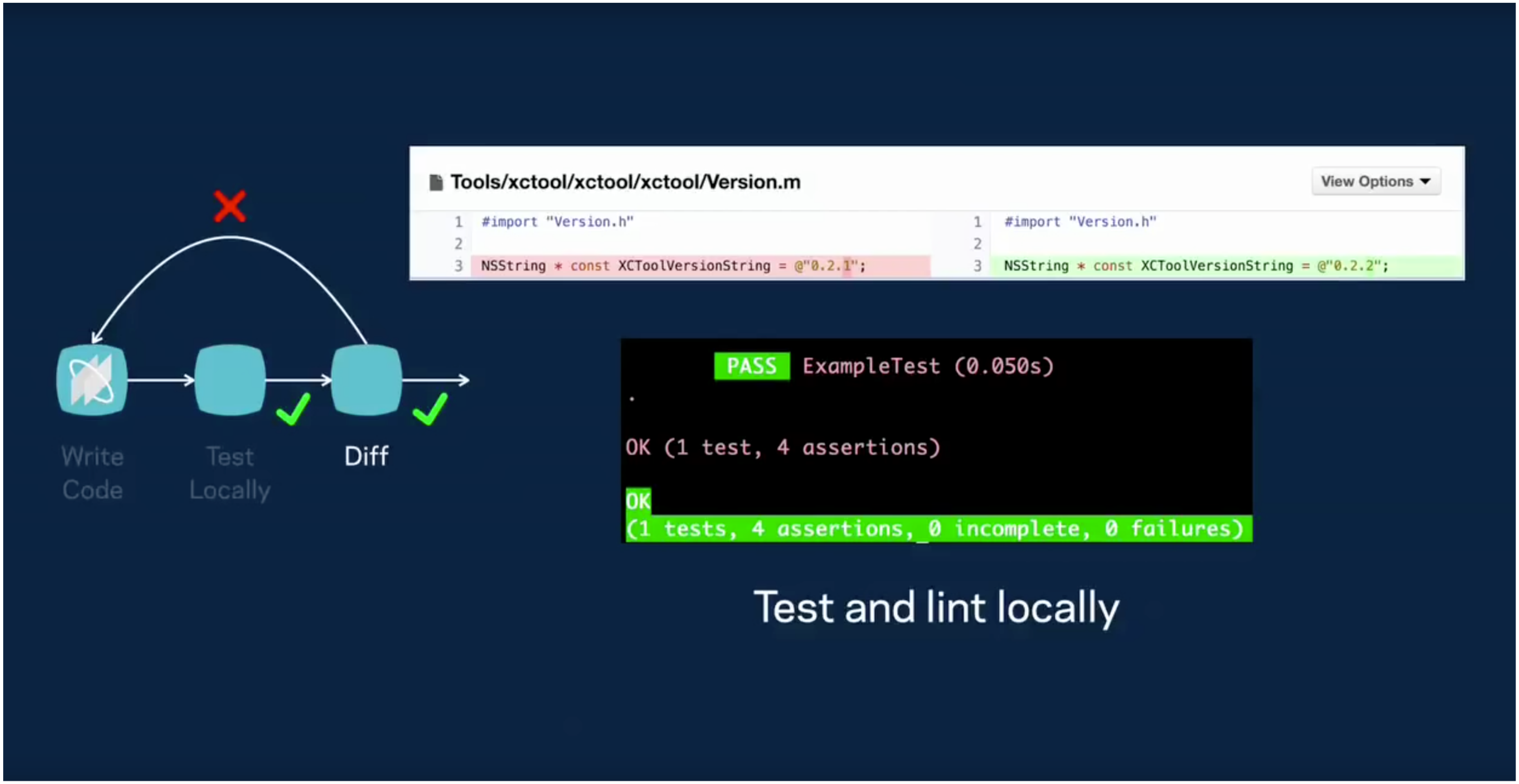
DEVELOPMENT AT SCALE

Releasing at scale in industry

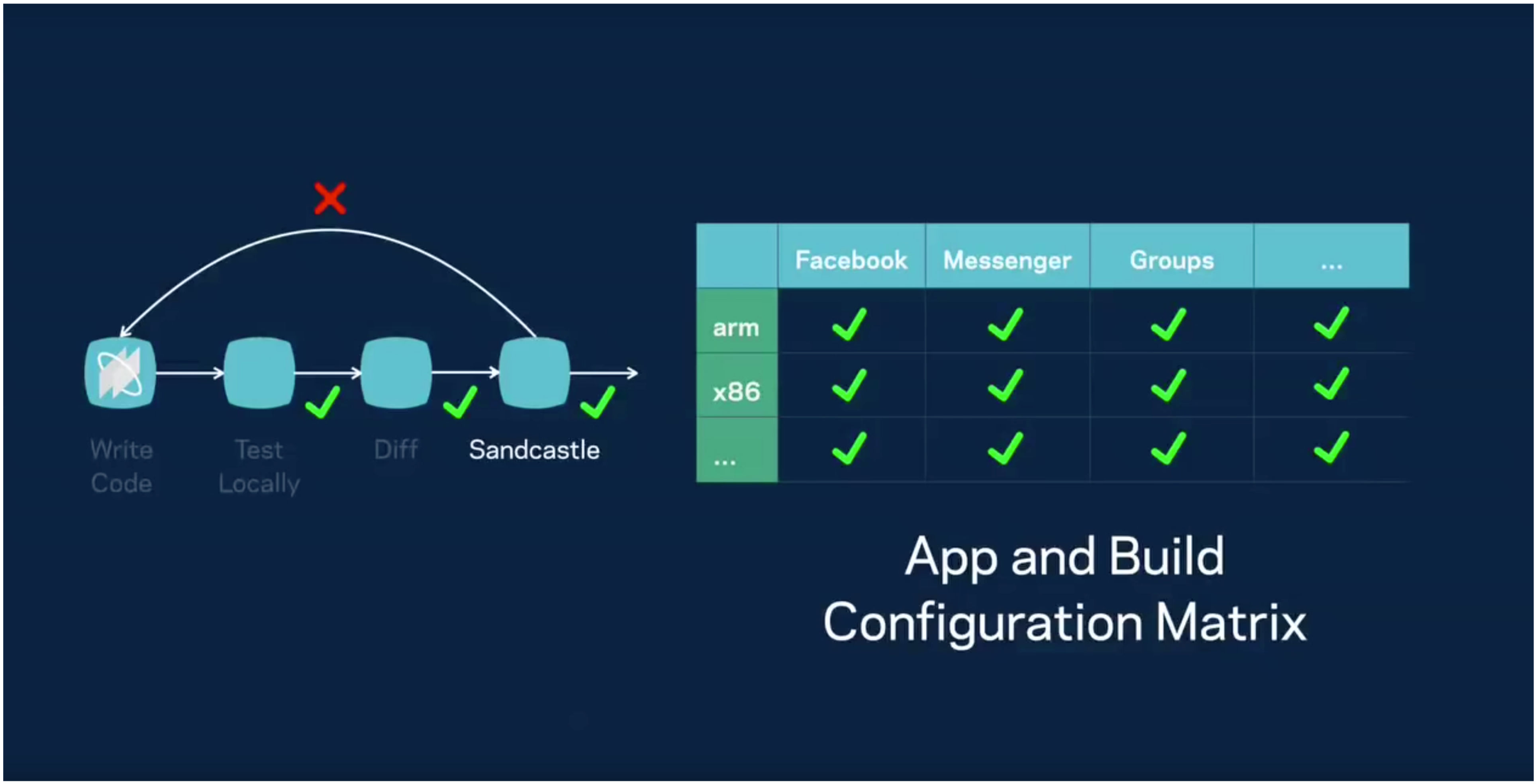
- Facebook: https://atscaleconference.com/videos/rapid-release-at-massive-scale/
- Google: https://www.slideshare.net/JohnMicco1/2016-0425-continuous-integration-at-google-scale
  - https://testing.googleblog.com/2011/06/testing-at-speed-and-scale-of-google.html
- Why Google Stores Billions of Lines of Code in a Single Repository: https://www.youtube.com/watch?v=W71BTkUbdqE
- F8 2015 - Big Code: Developer Infrastructure at Facebook's Scale: https://www.youtube.com/watch?v=X0VH78ye4yY

Pre-2017 release management model at Facebook

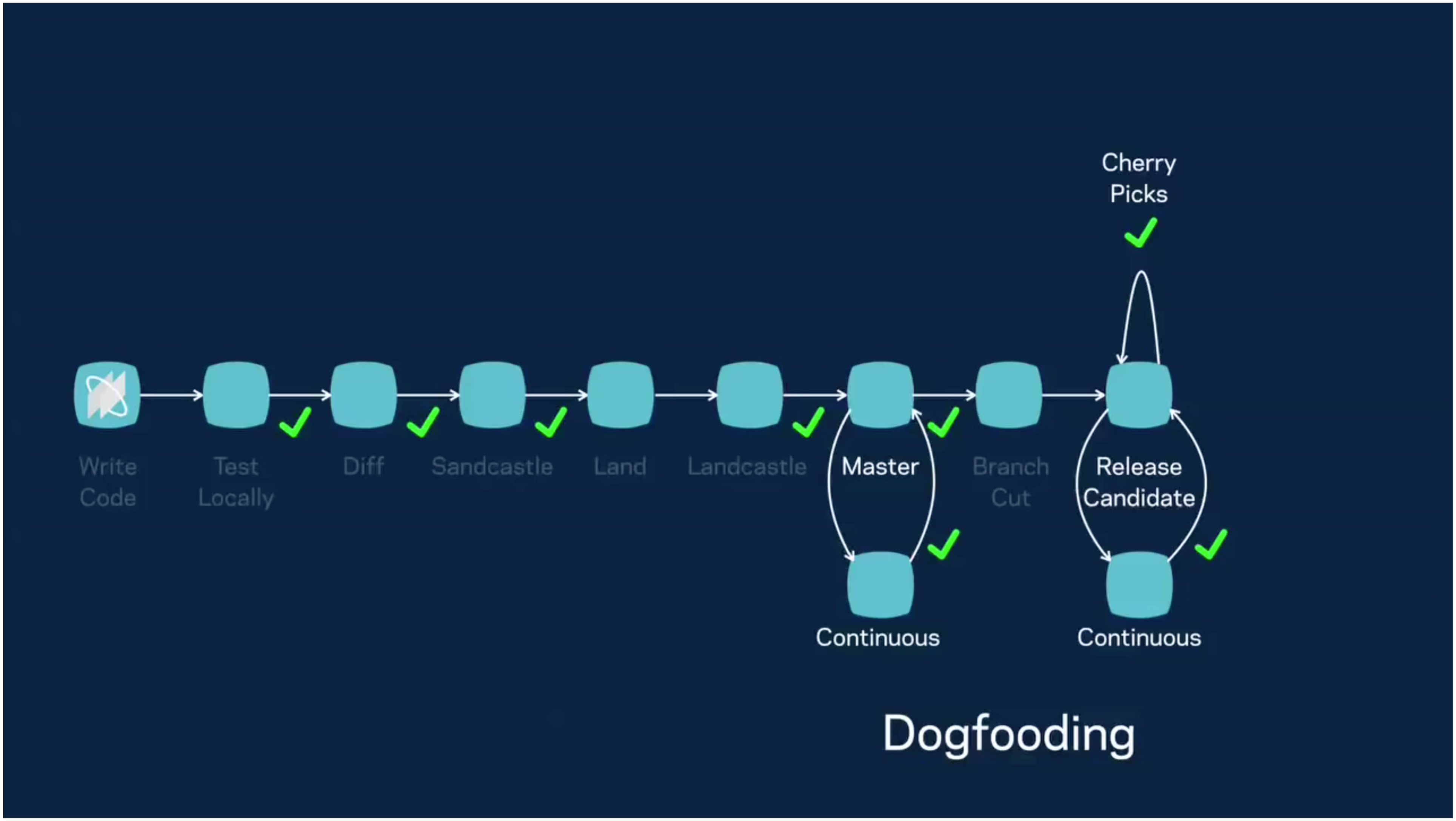
Diff lifecycle: local testing



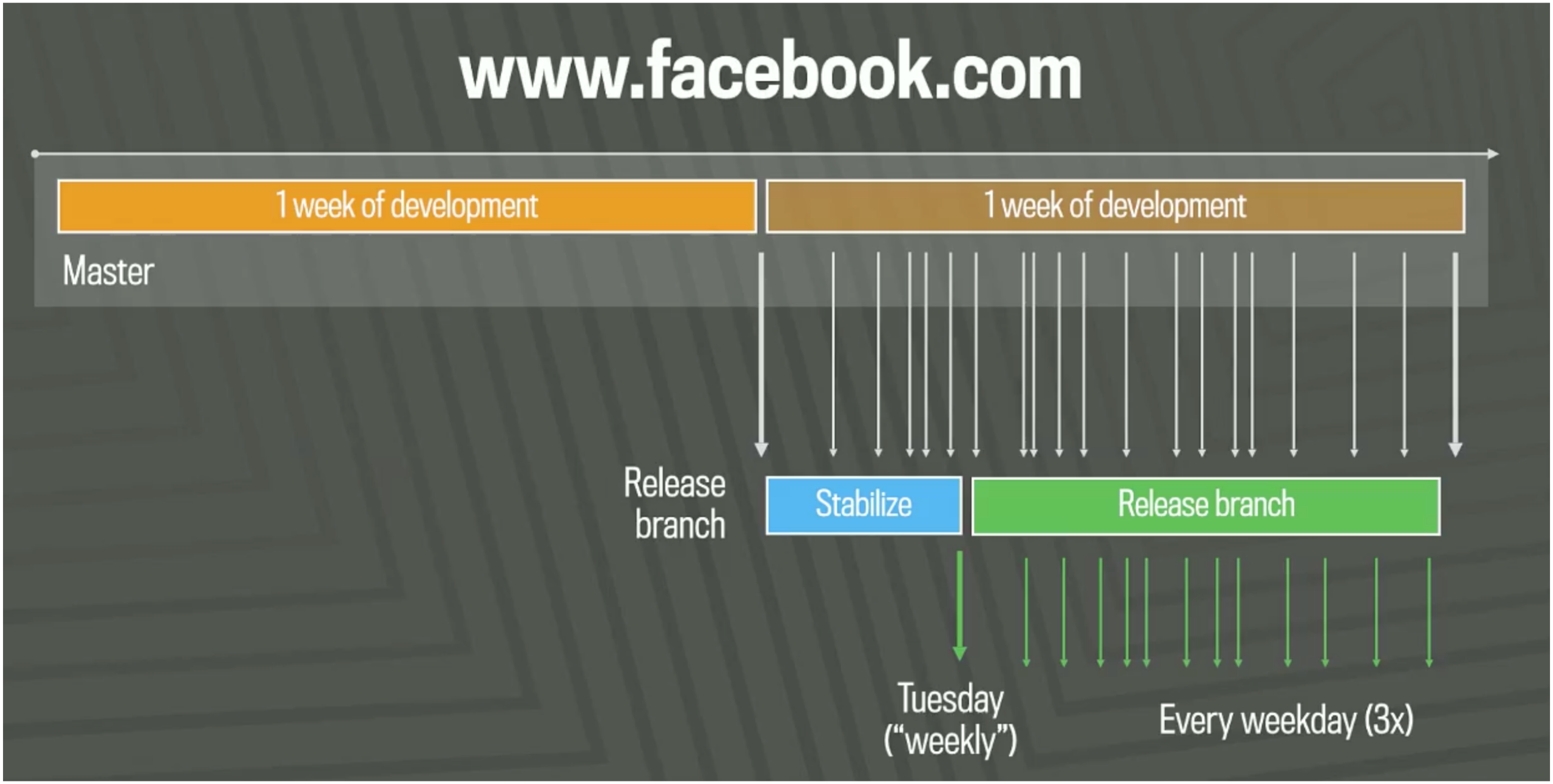
Diff lifecycle: CI testing (data center)



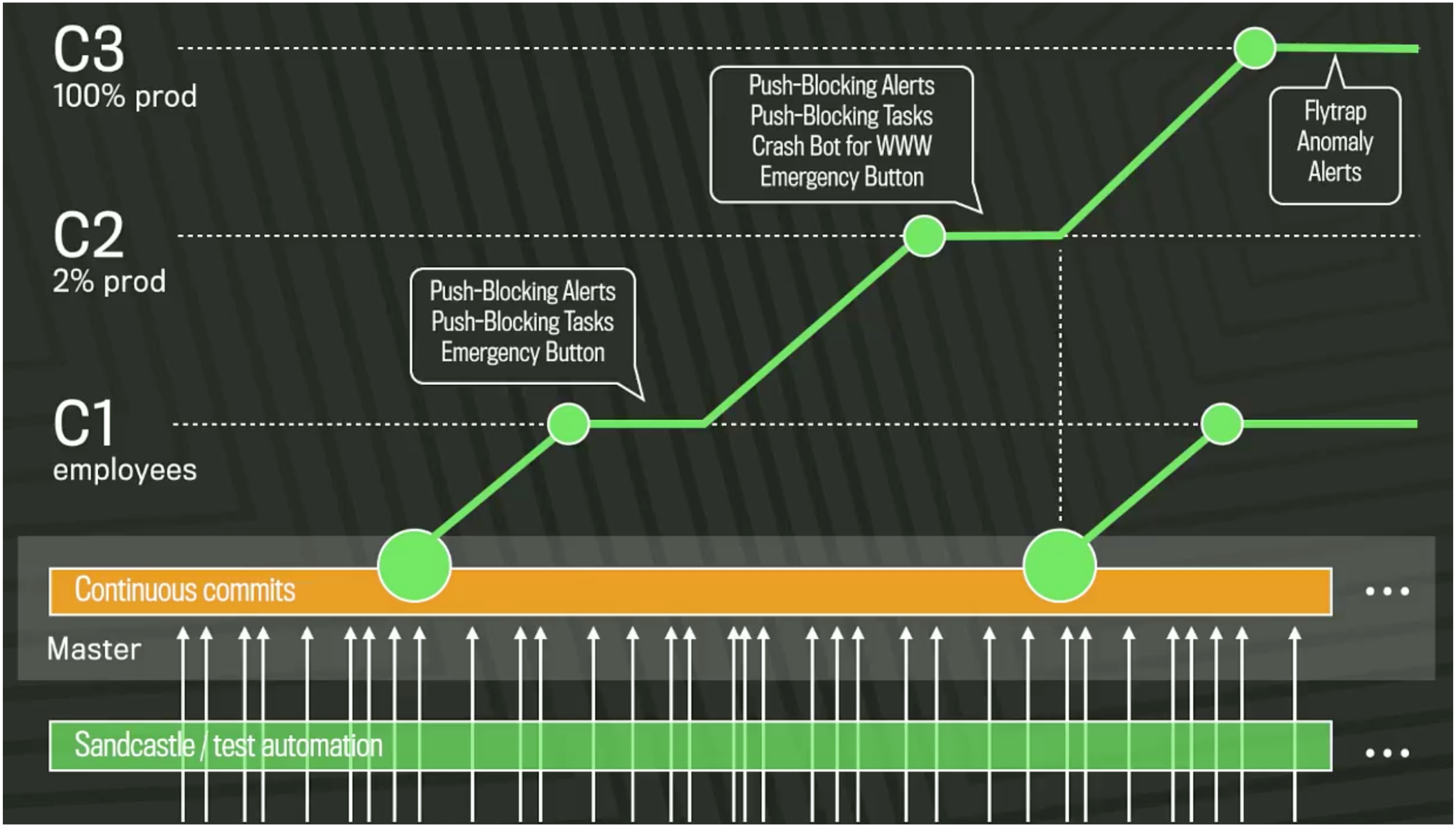
Diff lifecycle: diff ends up on master



Release every two weeks



Quasi-continuous push from master (1,000+ devs, 1,000 diffs/day); 10 pushes/day



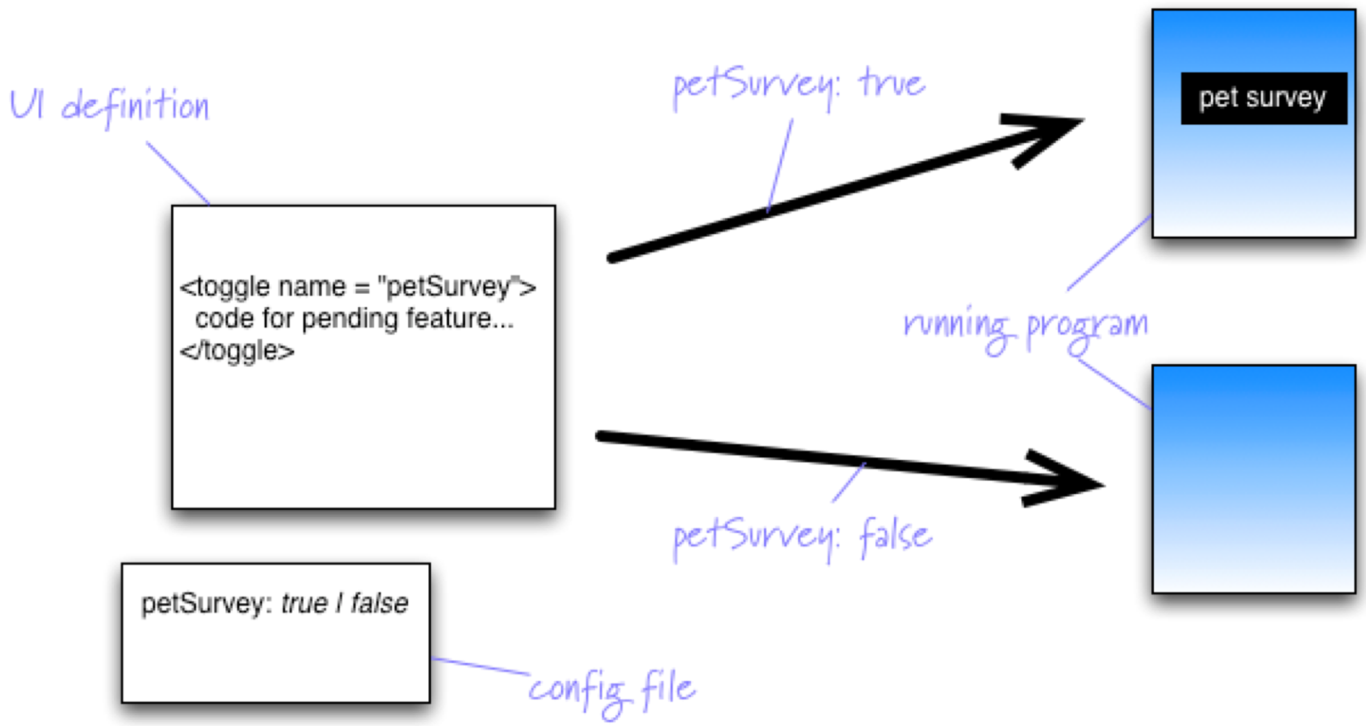
Aside: Key idea – fast to deploy, slow to release

Dark launches at Instagram

- **Early**: Integrate as soon as possible. Find bugs early. Code can run in production about 6 months before being publicly announced (“dark launch”).
- **Often**: Reduce friction. Try things out. See what works. Push small changes just to gather metrics, feasibility testing. Large changes just slow down the team. Do dark launches, to see what performance is in production, can scale up and down. *"Shadow infrastructure" is too expensive, just do in production.*
- **Incremental**: Deploy in increments. Contain risk. Pinpoint issues.

Aside: Feature Flags

Typical way to implement a dark launch.



http://swreflections.blogspot.com/2014/08/feature-toggles-are-one-of-worst-kinds.html

http://martinfowler.com/bliki/FeatureToggle.html

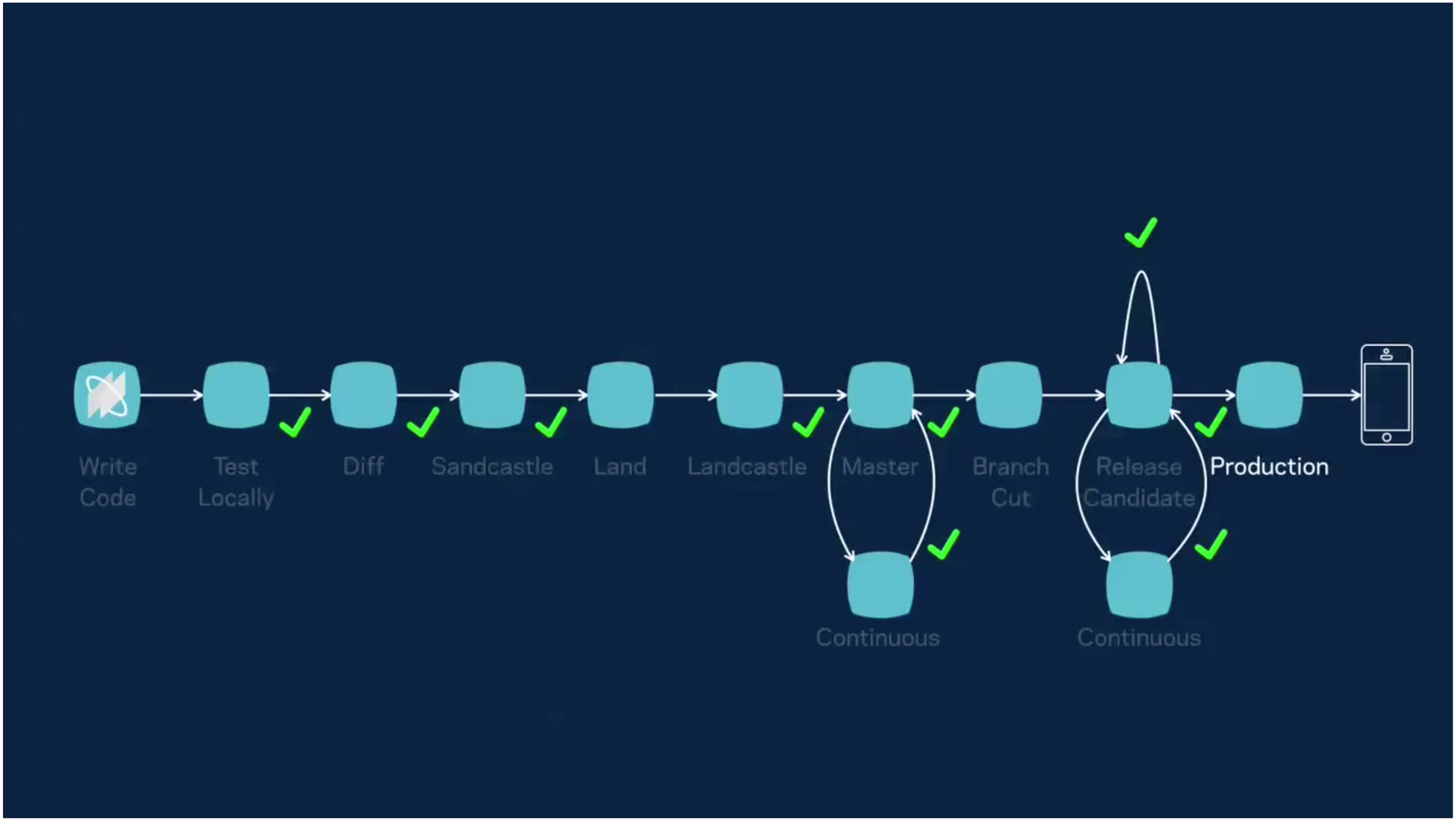
Issues with feature flags

Feature flags are "technical debt"

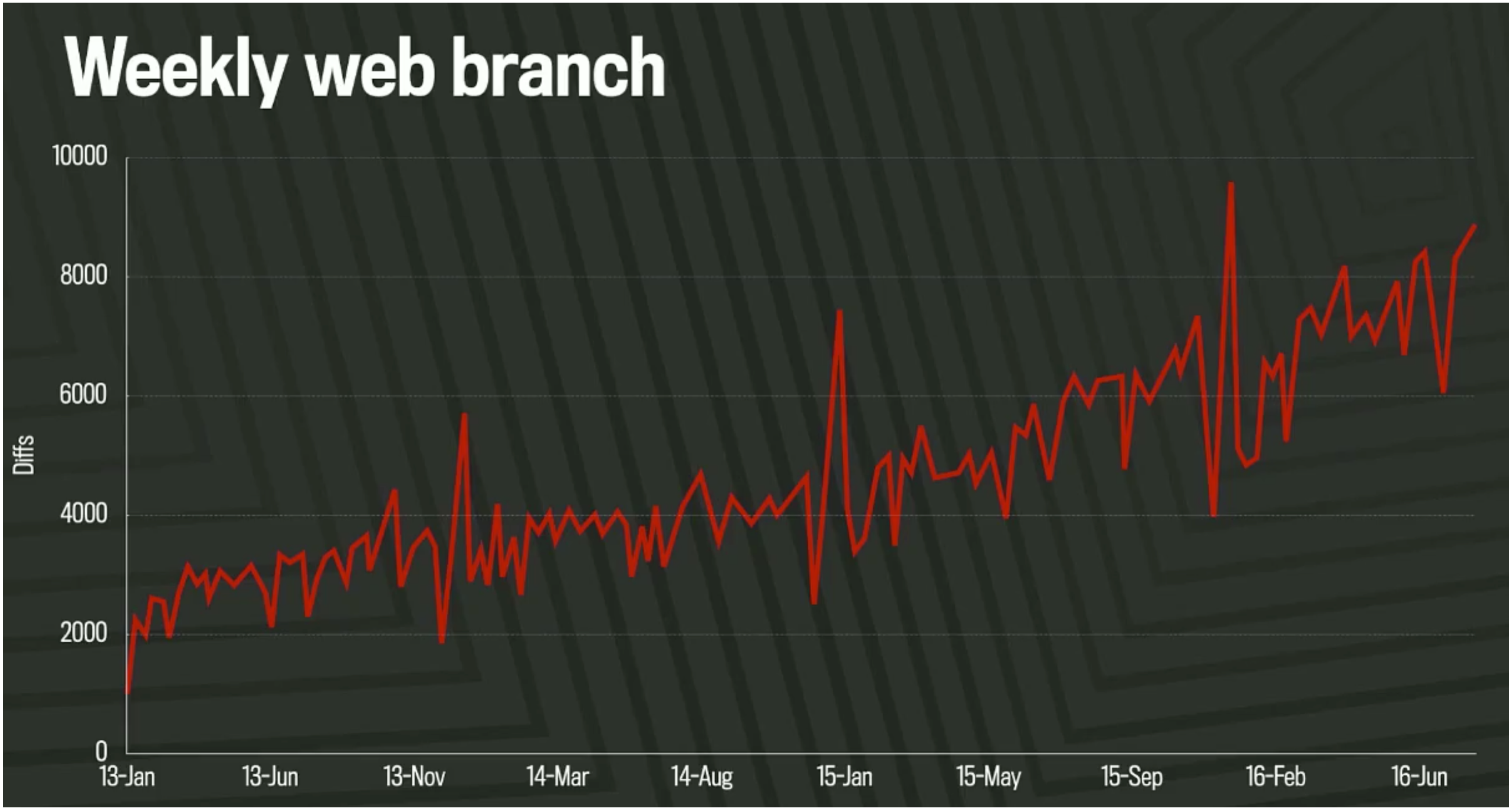
Example: financial services company went bankrupt in 45 minutes.

http://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/

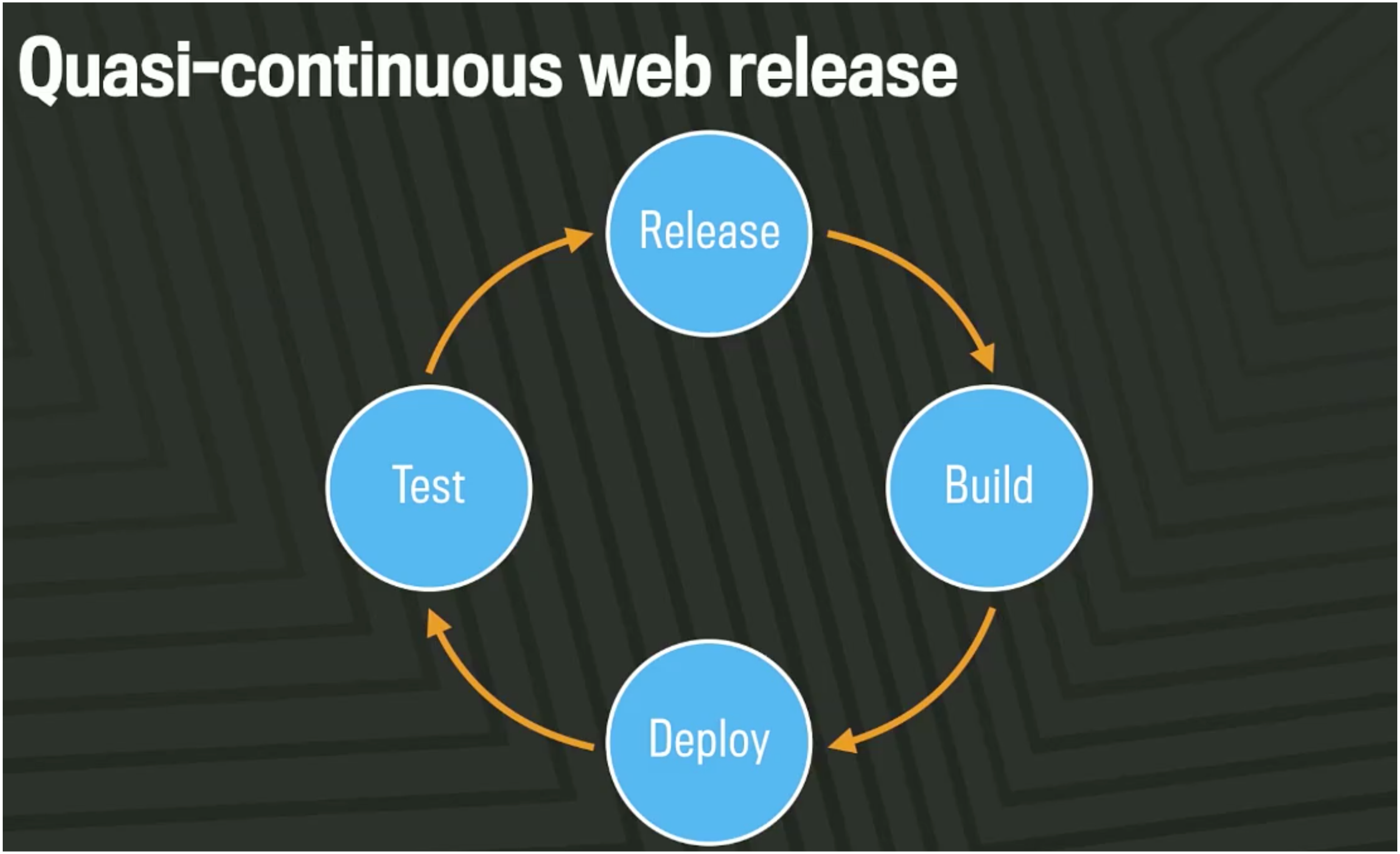
Diff lifecycle: in production



What's in a weekly branch cut? (The limits of branches)



Post-2017 release management model at Facebook



Google: similar story. HUGE code base

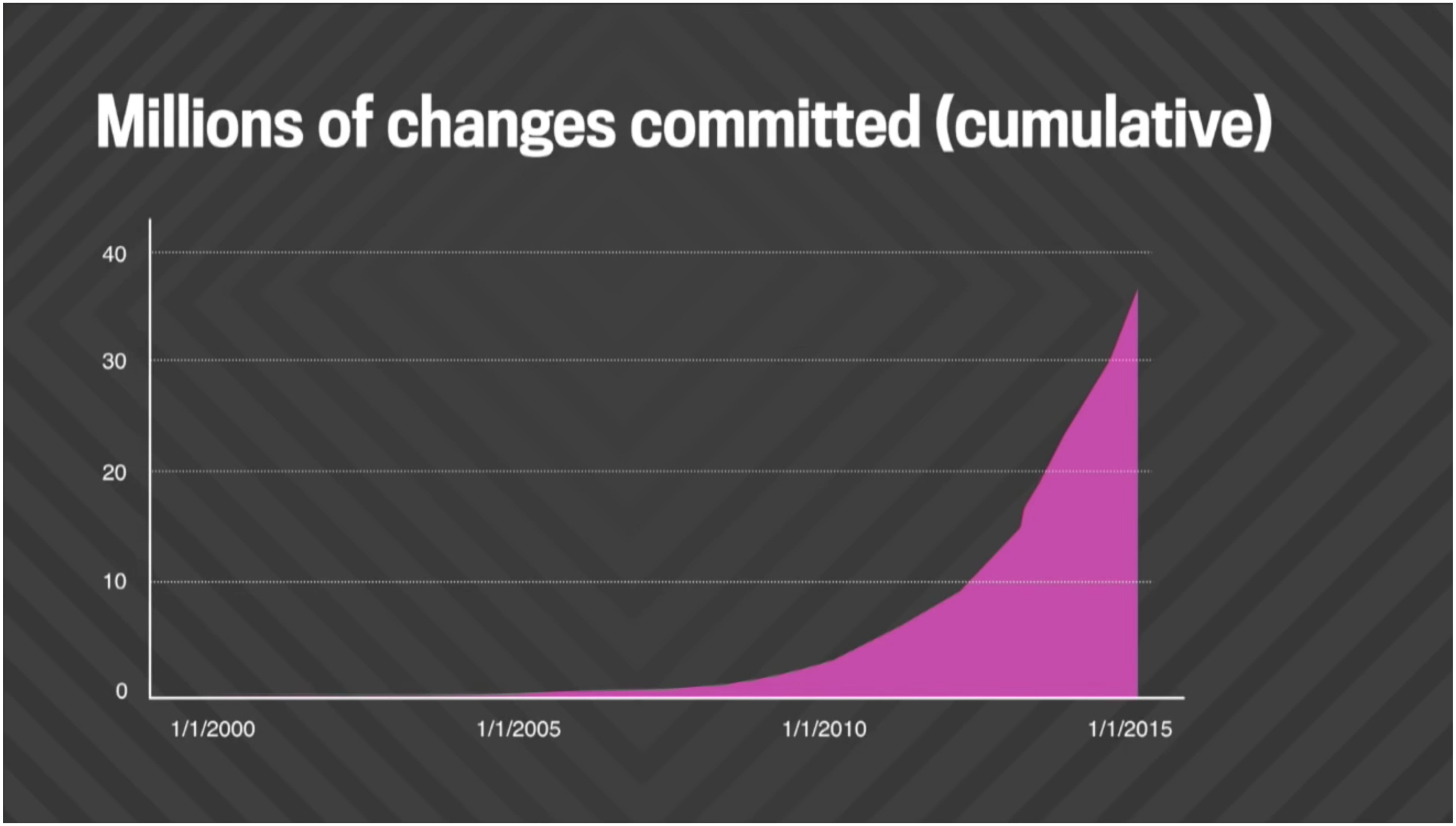
Google repository statistics

As of Jan 2015

Total number of files*	1 billion
Number of source files	9 million
Lines of code	2 billion
Depth of history	35 million commits
Size of content	86 terabytes
Commits per workday	45 thousand

*The total number of files includes source files copied into release branches, files that are deleted at the latest revision, configuration files, documentation, and supporting data files.

Exponential growth



2016 numbers

Google Speed and Scale

- >30,000 developers in 40+ offices
- 13,000+ projects under active development
- 30k submissions per day (1 every 3 seconds)

- All builds from source
- 30+ sustained code changes per minute with 90+ peaks
- 50% of code changes monthly
- 150+ million test cases / day, > 150 years of test / day
- Supports continuous deployment for all Google teams!

Google code base vs Linux kernel code base

Some perspective

Linux kernel

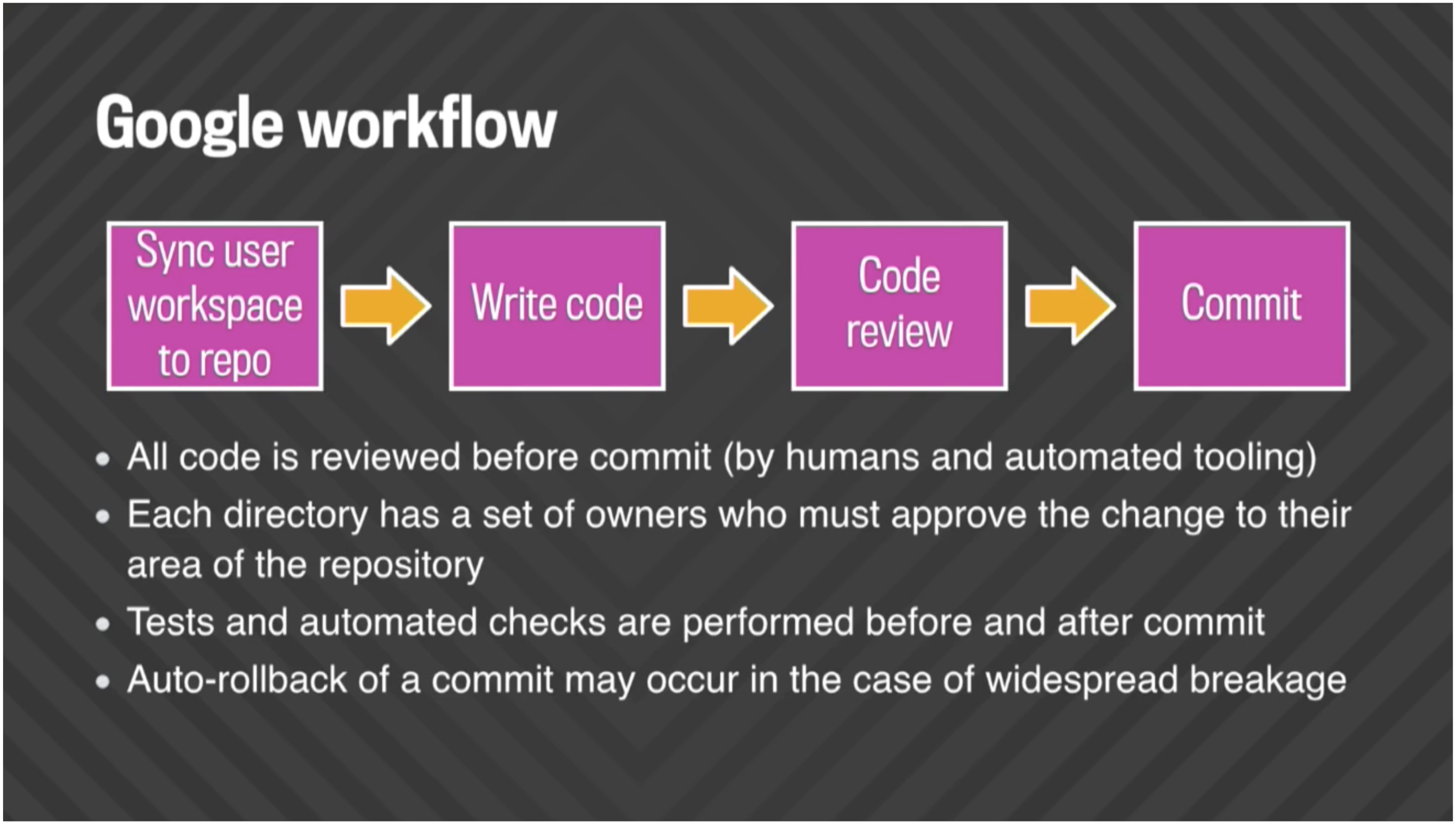
- 15 million lines of code in 40 thousand files (total)

Google repository

- 15 million lines of code in 250 thousand files *changed per week, by humans*
- 2 billion lines of code, in 9 million source files (total)

How do they do it?

1. Lots of (automated) testing



2. Lots of automation

Additional tooling support

Critique	Code review
CodeSearch*	Code browsing, exploration, understanding, and archeology
Tricorder**	Static analysis of code surfaced in Critique, CodeSearch
Presubmits	Customizable checks, testing, can block commit
TAP	Comprehensive testing before and after commit, auto-rollback
Rosie	Large-scale change distribution and management

* See "How Developers Search for Code: A Case Study", In European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2015

** See "Tricorder: Building a program analysis ecosystem". In International Conference on Software Engineering (ICSE), 2015

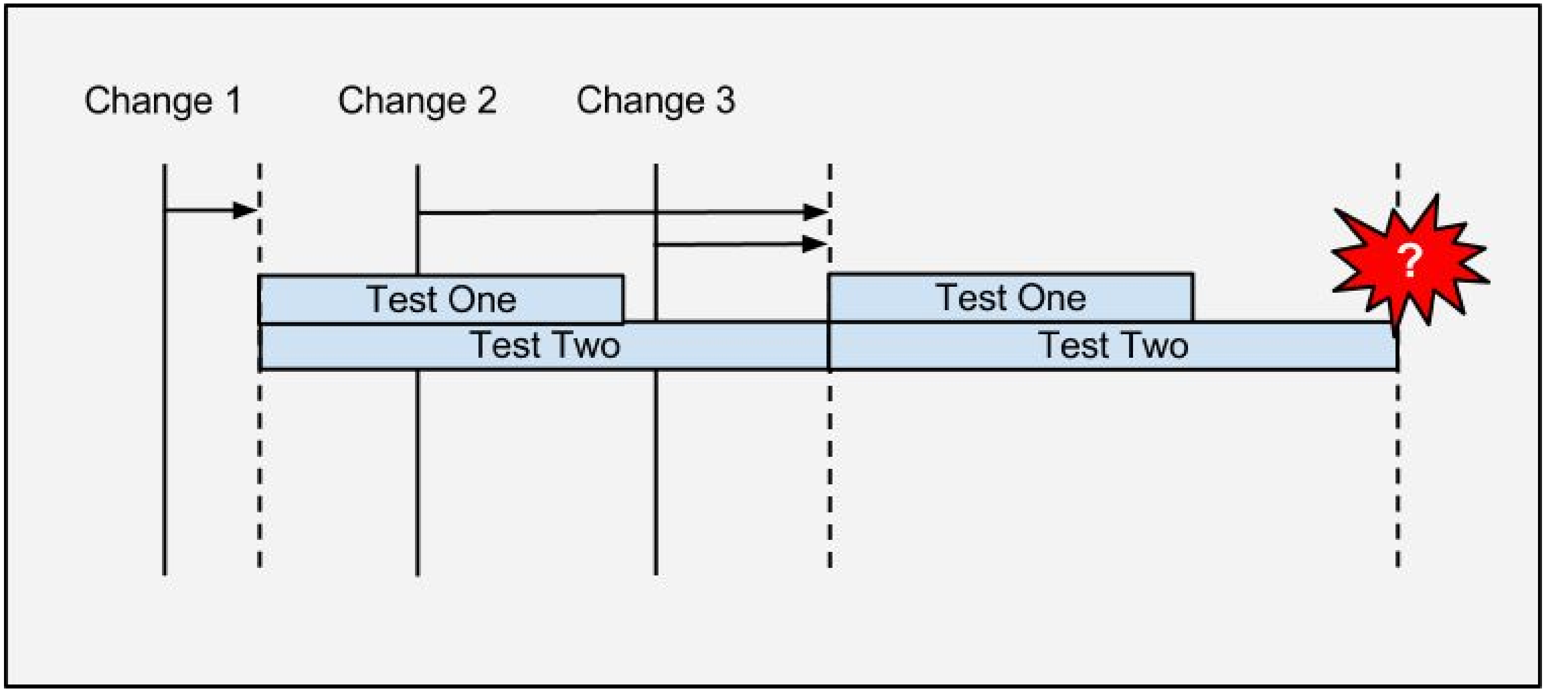
3. Smarter tooling

- Build system
- Version control
- …

3a. Build system

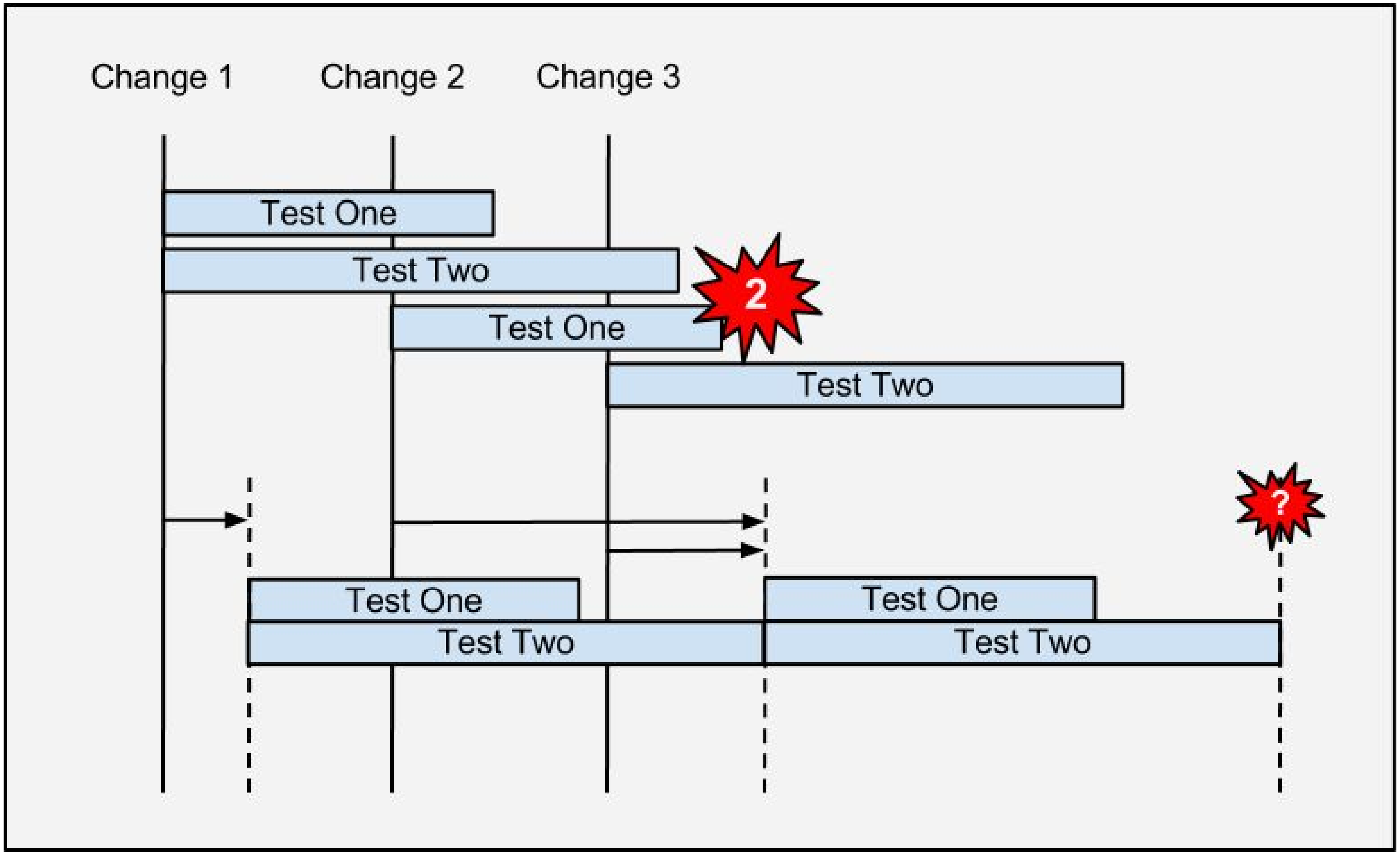
Google Standard Continuous Build System

- Triggers builds in continuous cycle
- Cycle time = longest build + test cycle
- Tests many changes together
- Which change broke the build?

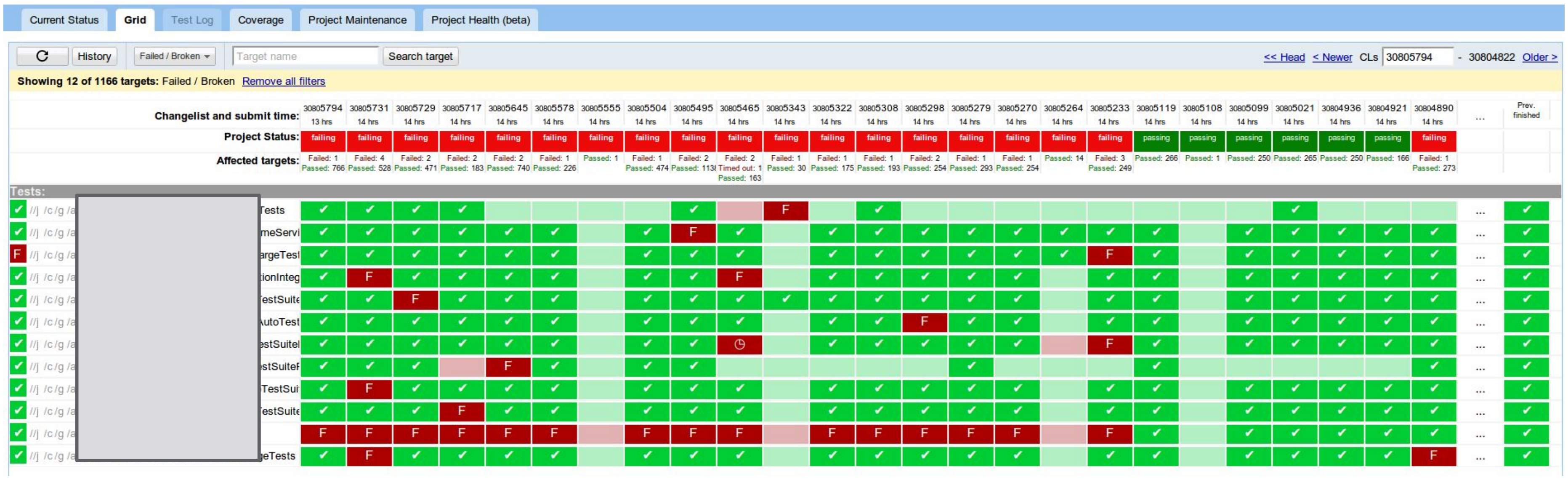


Google Continuous Build System

- Triggers tests on every change
- Uses fine-grained dependencies
- Change 2 broke test 1



Google Continuous Integration Display



Google Confidential and Proprietary

ISI SOFTWARE RESEARCH

Google Benefits

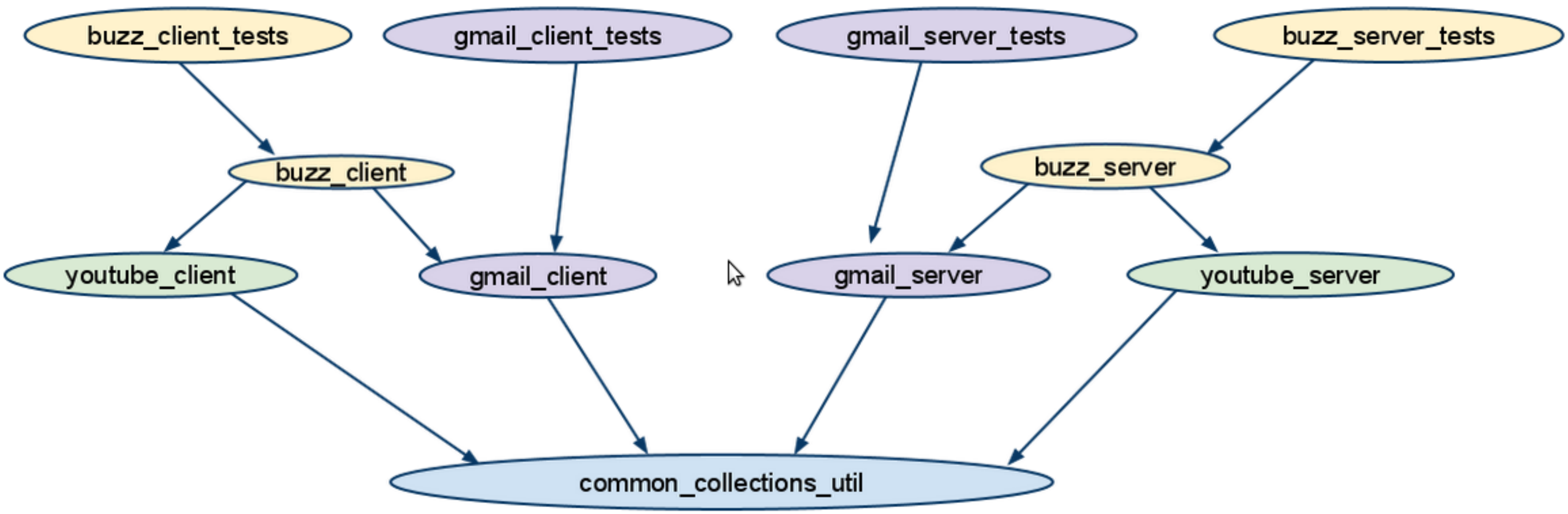
- Identifies failures sooner
- Identifies culprit change precisely
  - Avoids divide-and-conquer and tribal knowledge
- Lower compute costs using fine grained dependencies
- Keeps the build green by reducing time to fix breaks
- Accepted enthusiastically by product teams
- Enables teams to ship with fast iteration times
  - Supports submit-to-production times of less than 36 hours for some projects

Google Costs

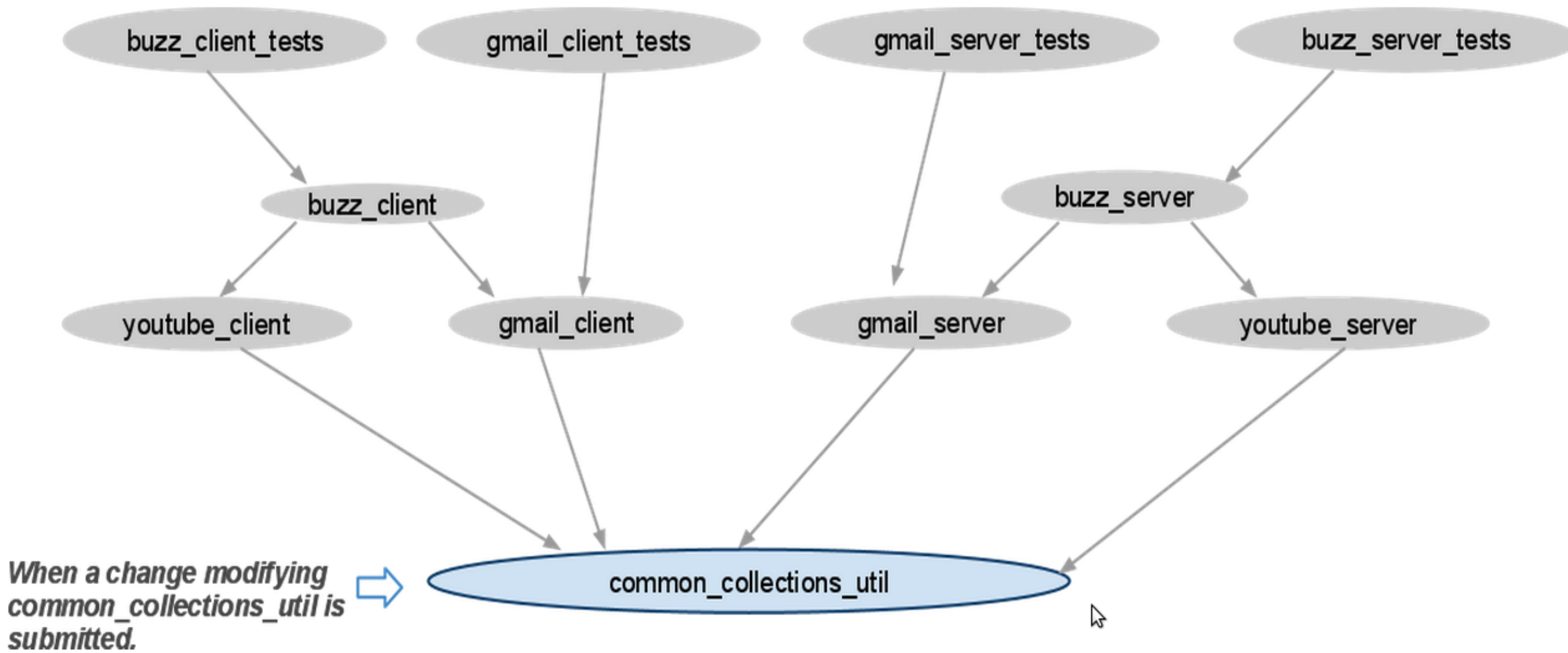
- Requires enormous investment in compute resources (it helps to be at Google) grows in proportion to:
  - Submission rate
  - Average build + test time
  - Variants (debug, opt, valgrind, etc.)
  - Increasing dependencies on core libraries
  - Branches
- Requires updating dependencies on each change
  - Takes time to update - delays start of testing

Which tests to run?

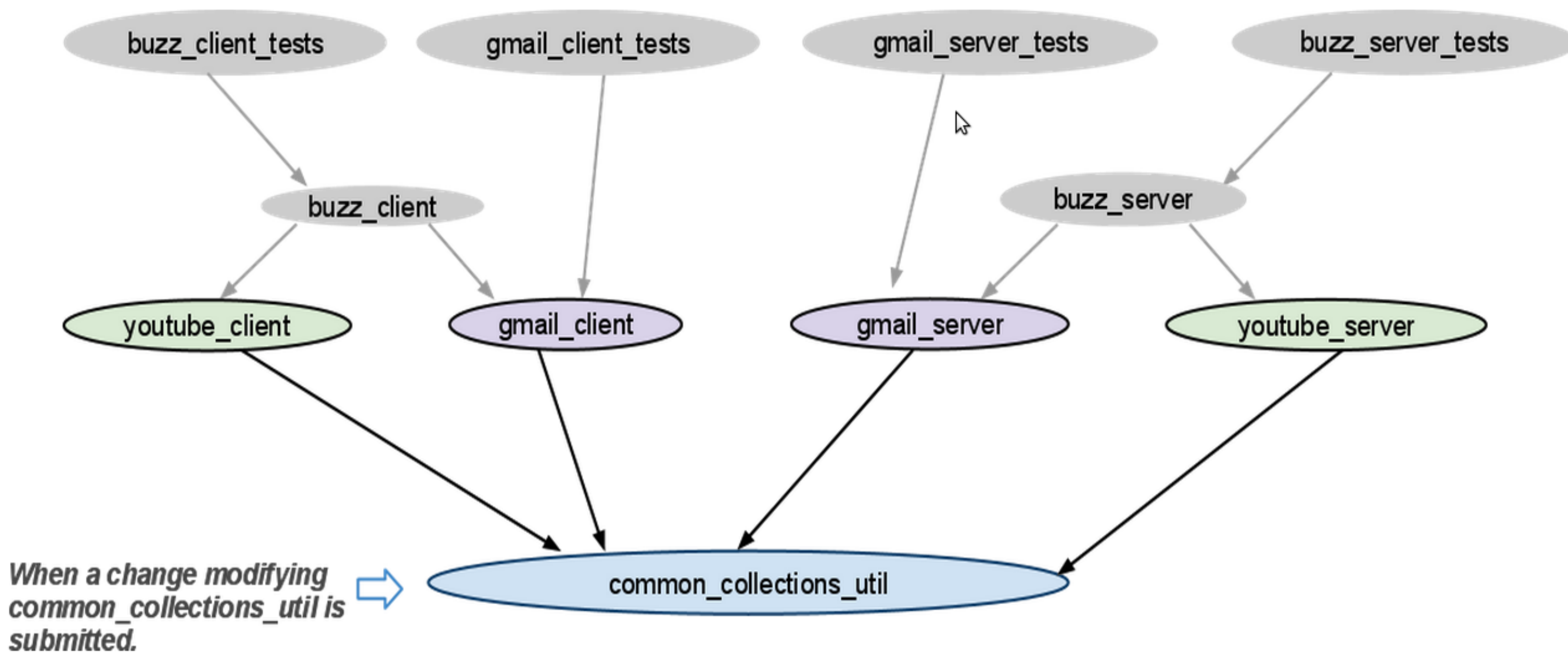
GMAIL	BUZZ
Test Target: name: //depot/gmail_client_tests name: //depot/gmail_server_tests	Test targets: name: //depot/buzz_server_tests name: //depot/buzz_client_tests



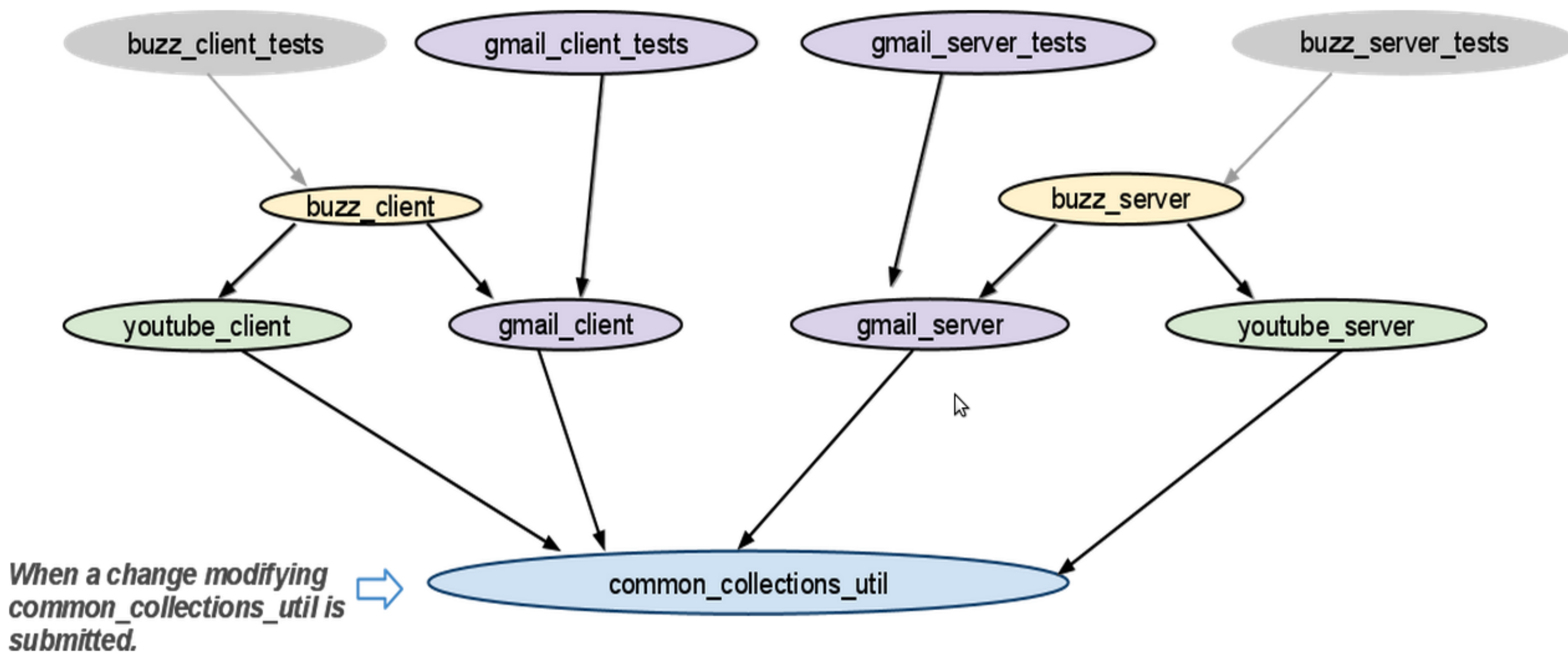
Scenario 1: a change modifies common_collections_util



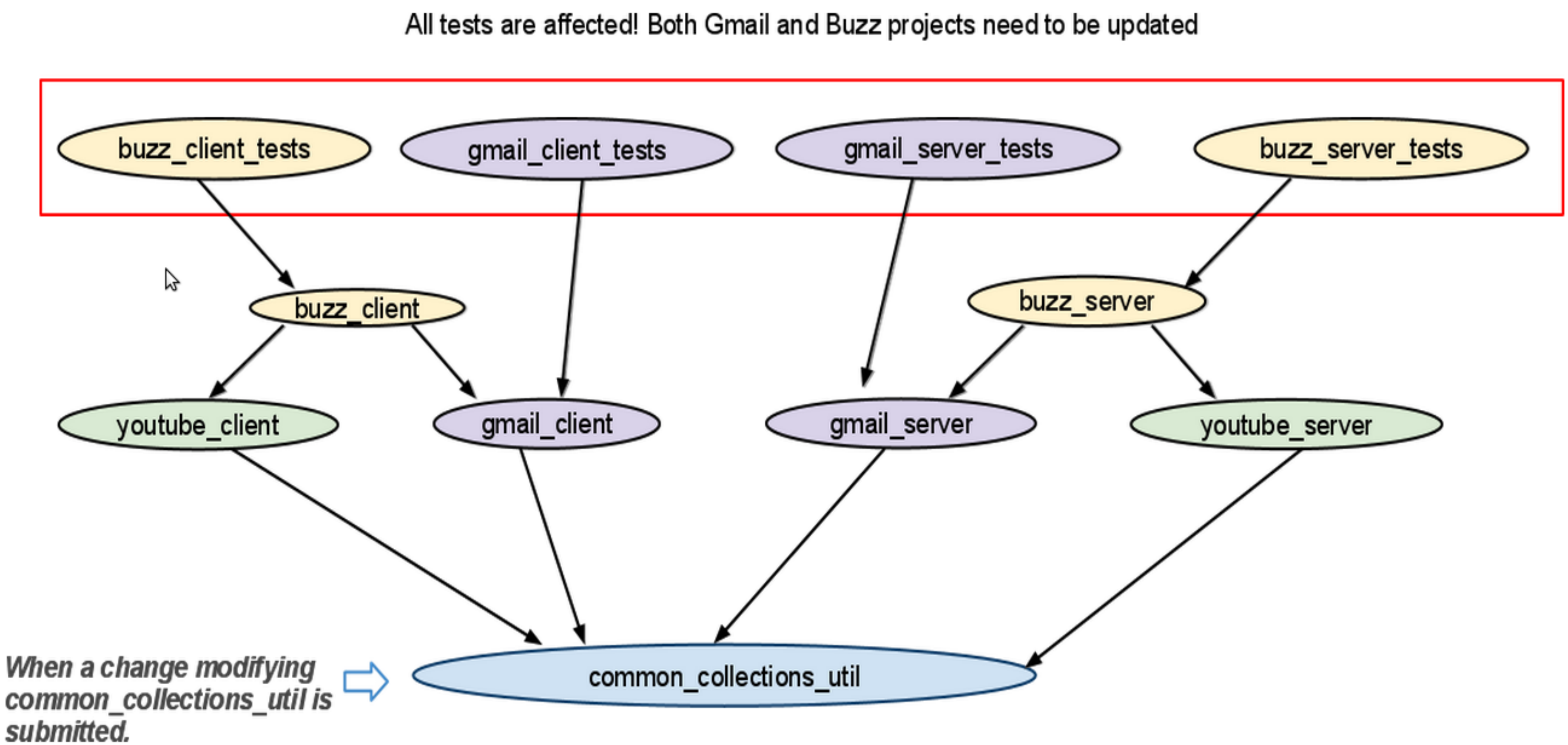
Scenario 1: a change modifies common_collections_util



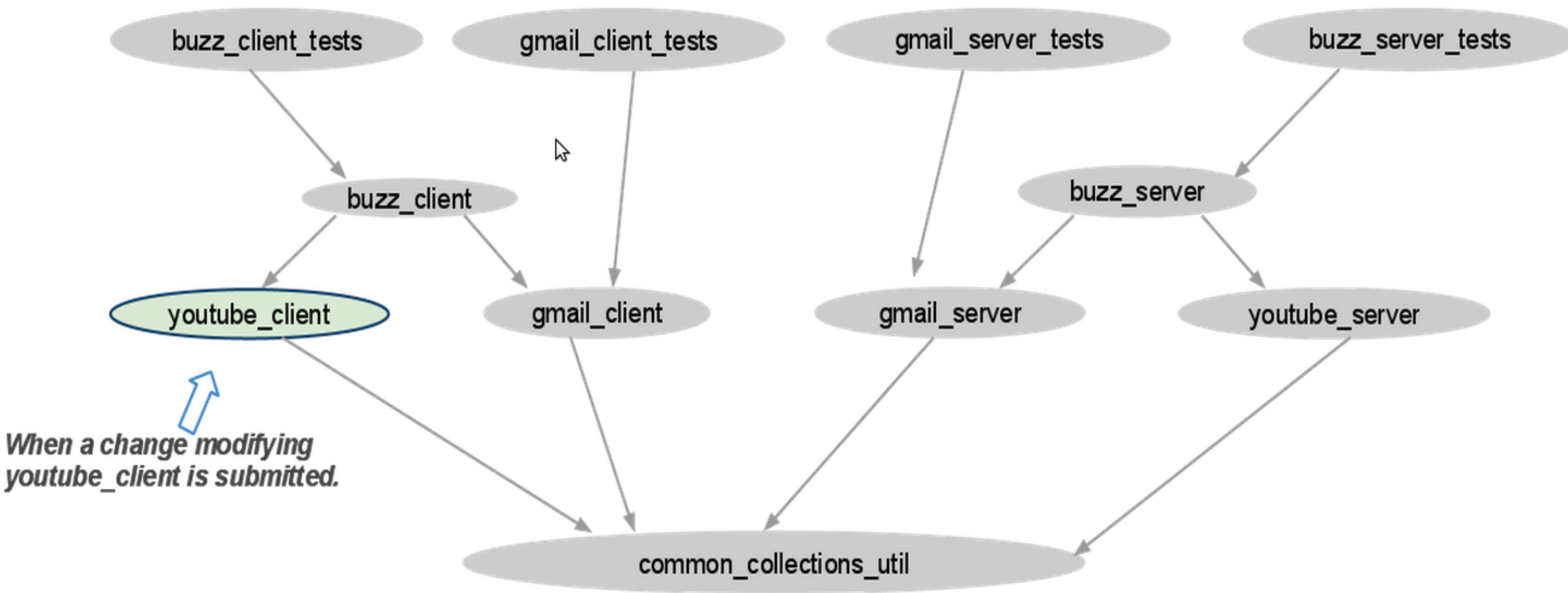
Scenario 1: a change modifies common_collections_util



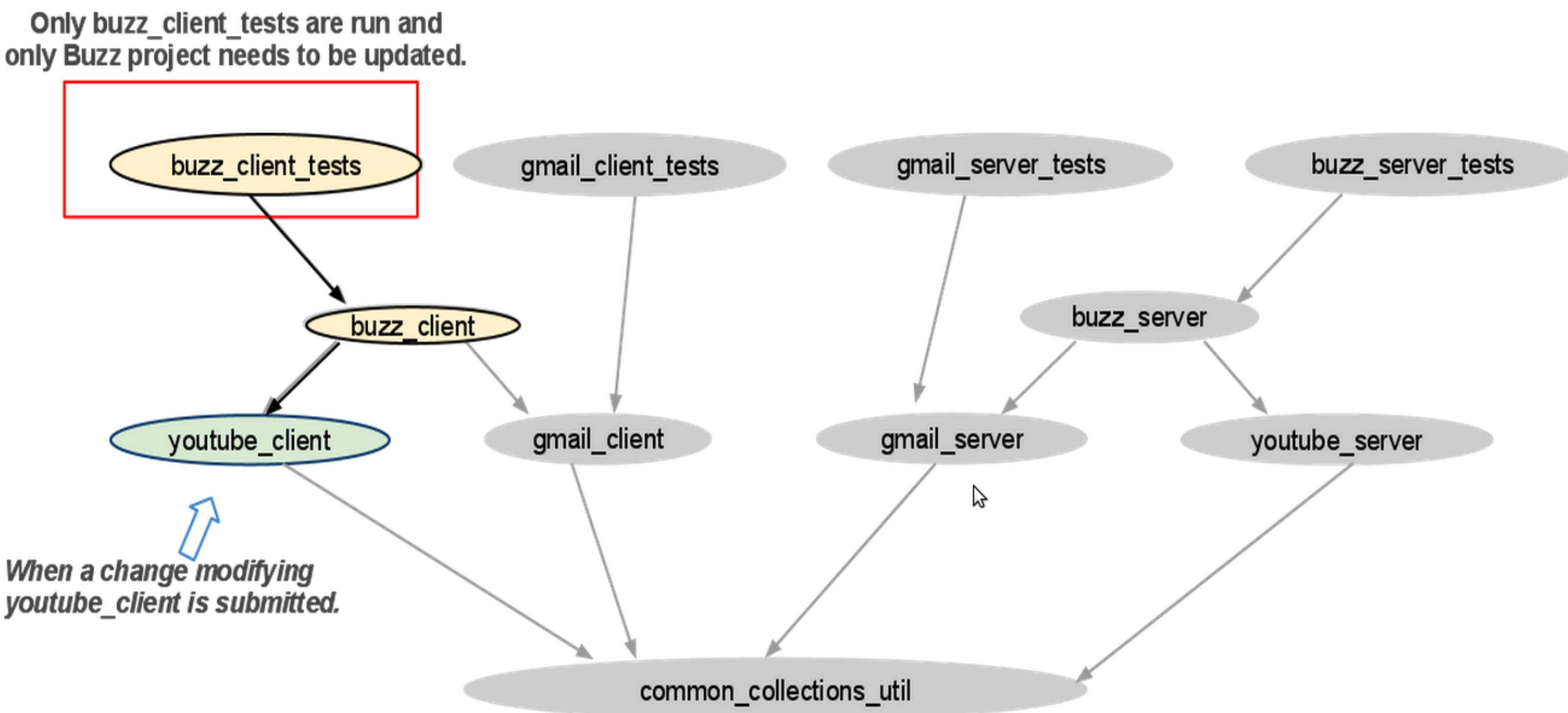
Scenario 1: a change modifies common_collections_util



Scenario 2: a change modifies the youtube_client

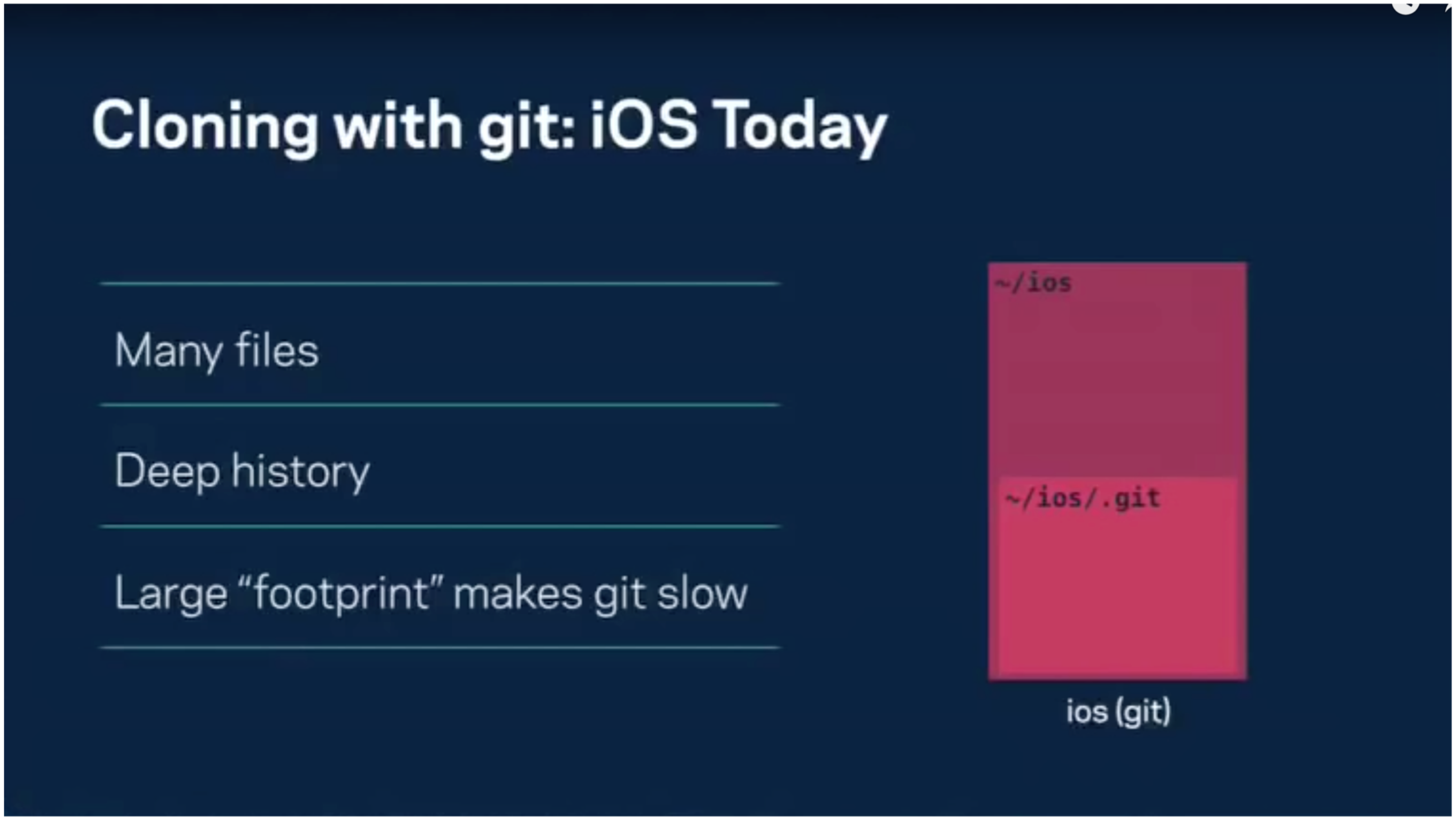


Scenario 2: a change modifies the youtube_client



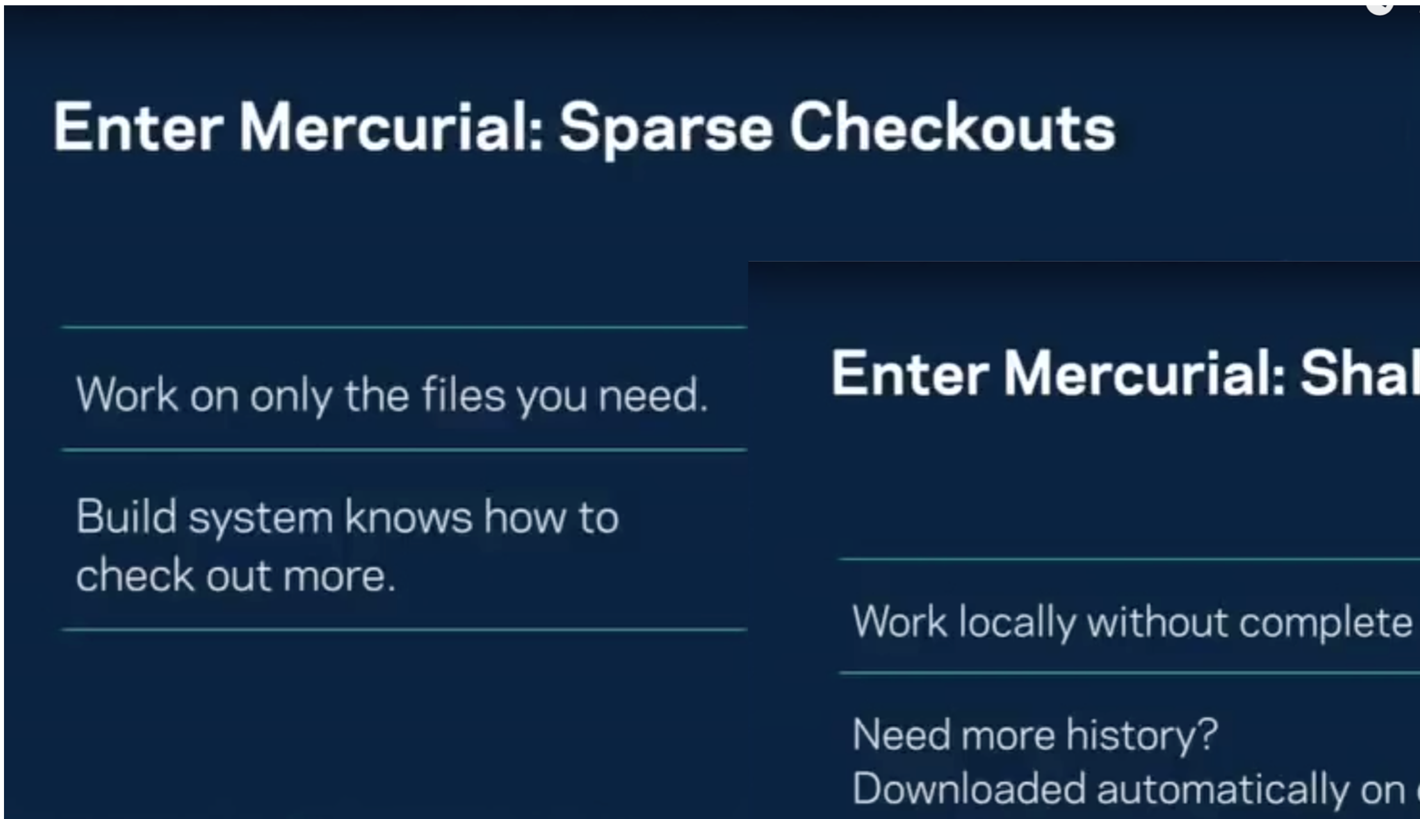
3b. Version control

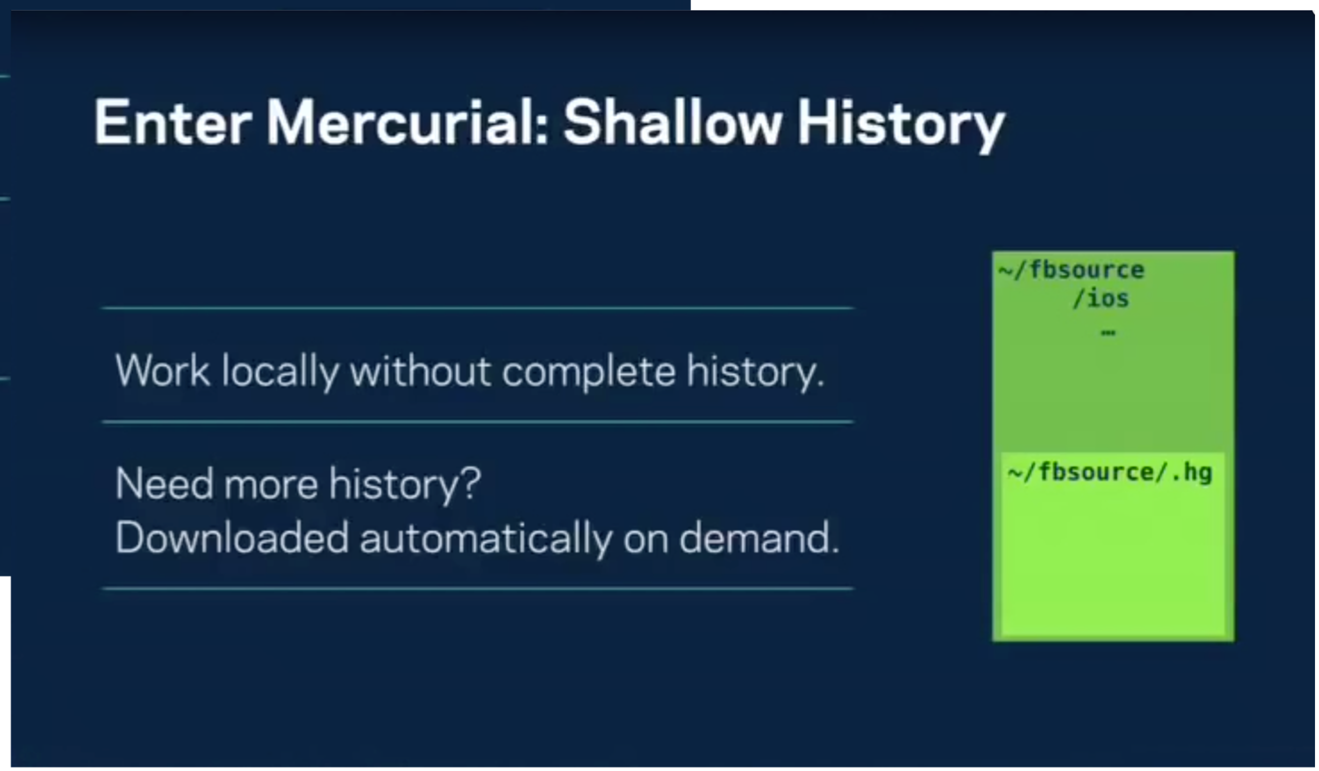
- Problem: even git can get slow at Facebook scale
  - 1M+ source control commands run per day
  - 100K+ commits per week



3b. Version control

- Solution: redesign version control



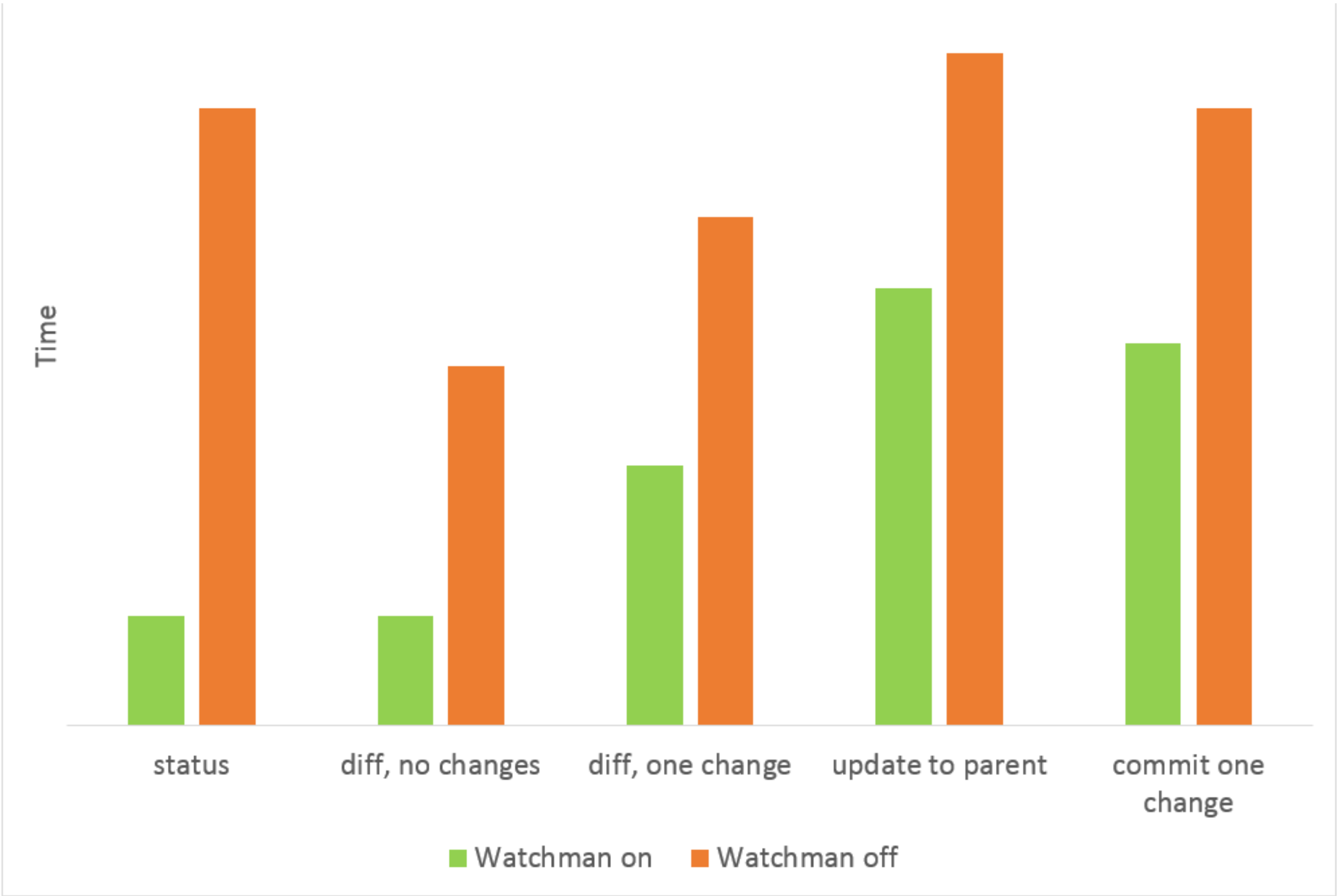


3b. Version control

- Solution: redesign version control
  - Query build system's file monitor, Watchman, to see which files have changed

3b. Version control

- Solution: redesign version control
  - Query build system's file monitor, Watchman, to see which files have changed → **5x faster "status" command**



3b. Version control

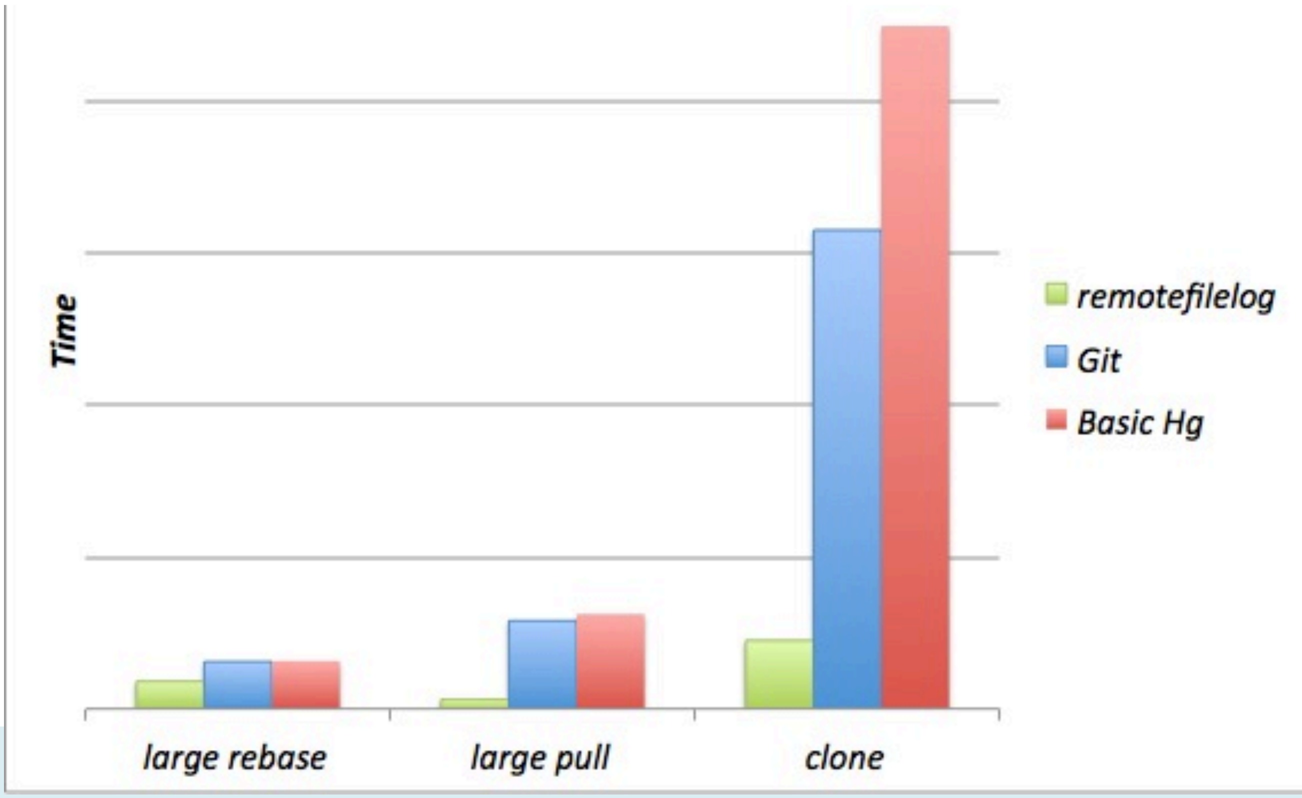
- Solution: redesign version control
  - Sparse checkouts??? (remember, git is a distributed VCS)

3b. Version control

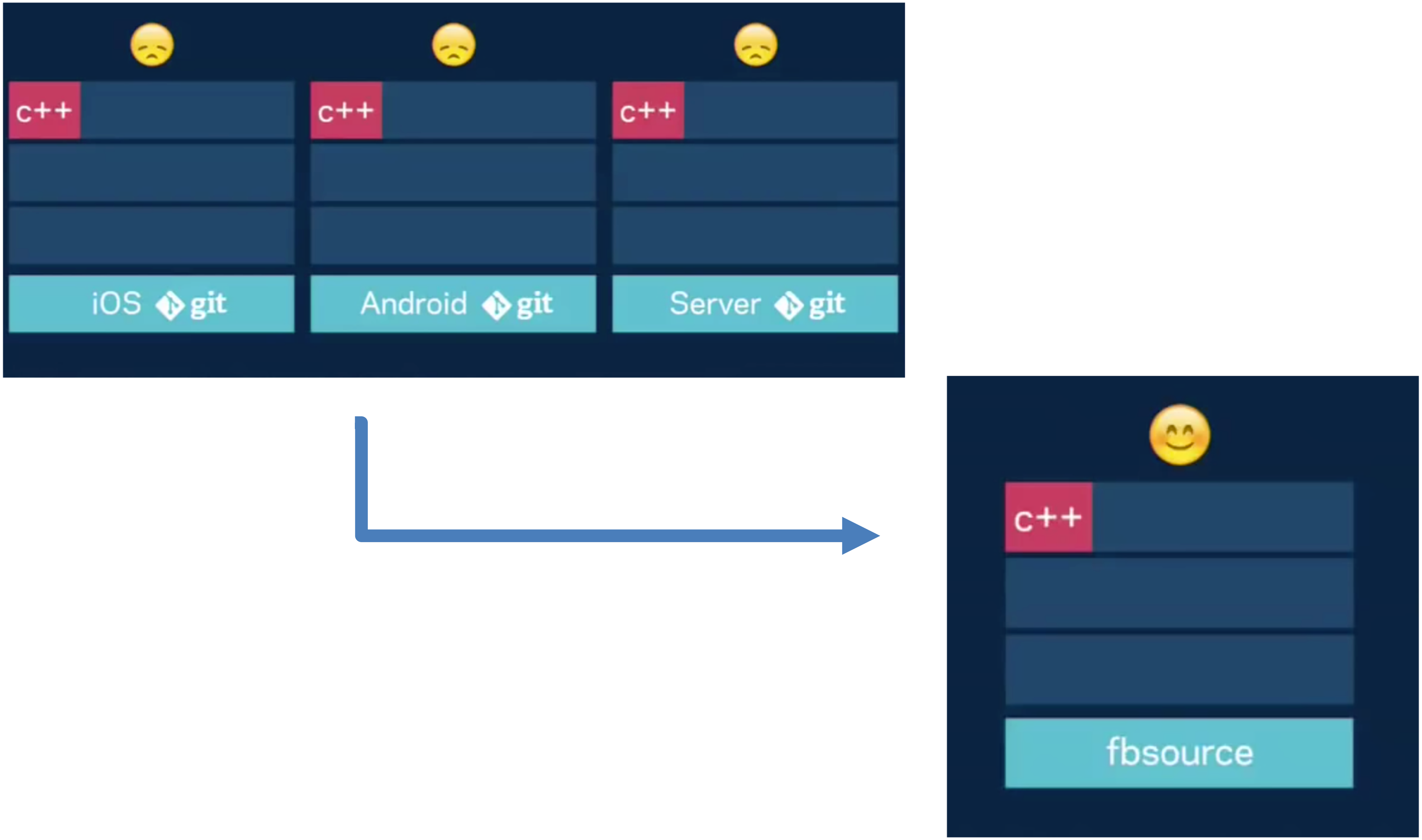
- Solution: redesign version control
  - Sparse checkouts:
  - Change the clone and pull commands to download only the commit metadata, while omitting all file changes (the bulk of the download)
  - When a user performs an operation that needs the contents of files (such as checkout), download the file contents on demand using existing memcache infrastructure

3b. Version control

- Solution: redesign version control
  - Sparse checkouts → **10x faster clones and pulls**
  - Change the clone and pull commands to download only the commit metadata, while omitting all file changes (the bulk of the download)
  - When a user performs an operation that needs the contents of files (such as checkout), download the file contents on demand using existing memcache infrastructure



4. Monolithic repository

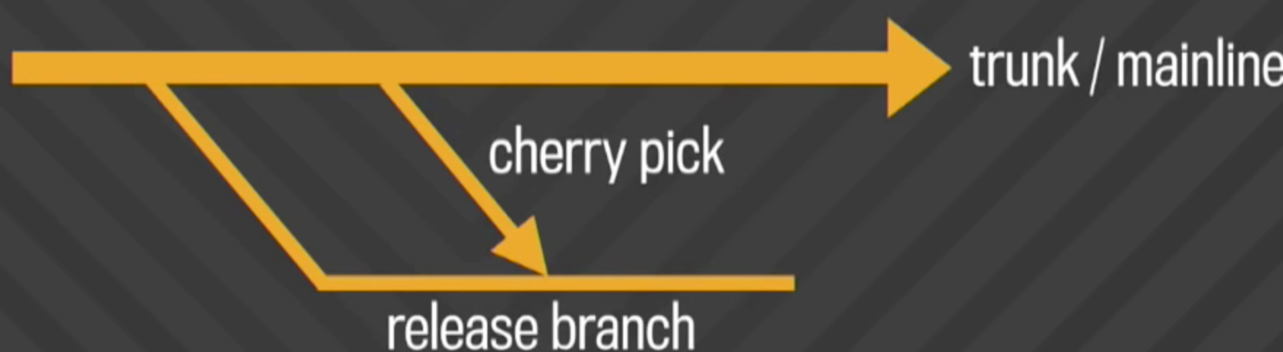


Monolithic repository – no major use of branches for development

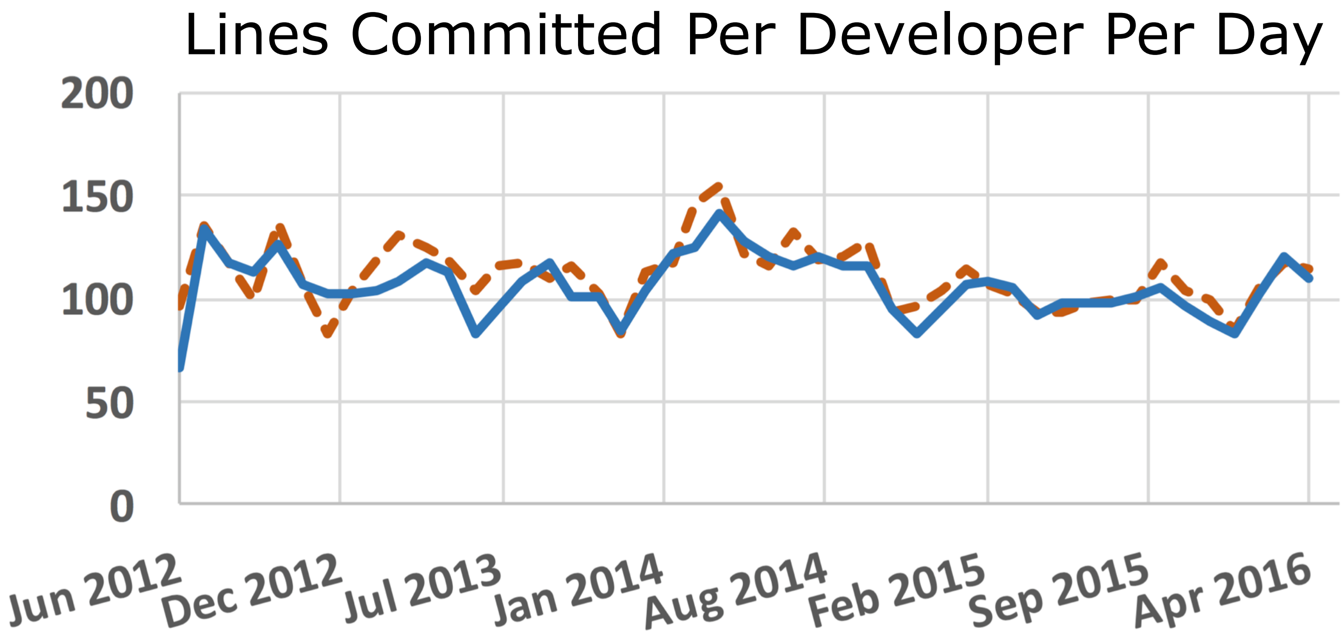
Trunk-based development

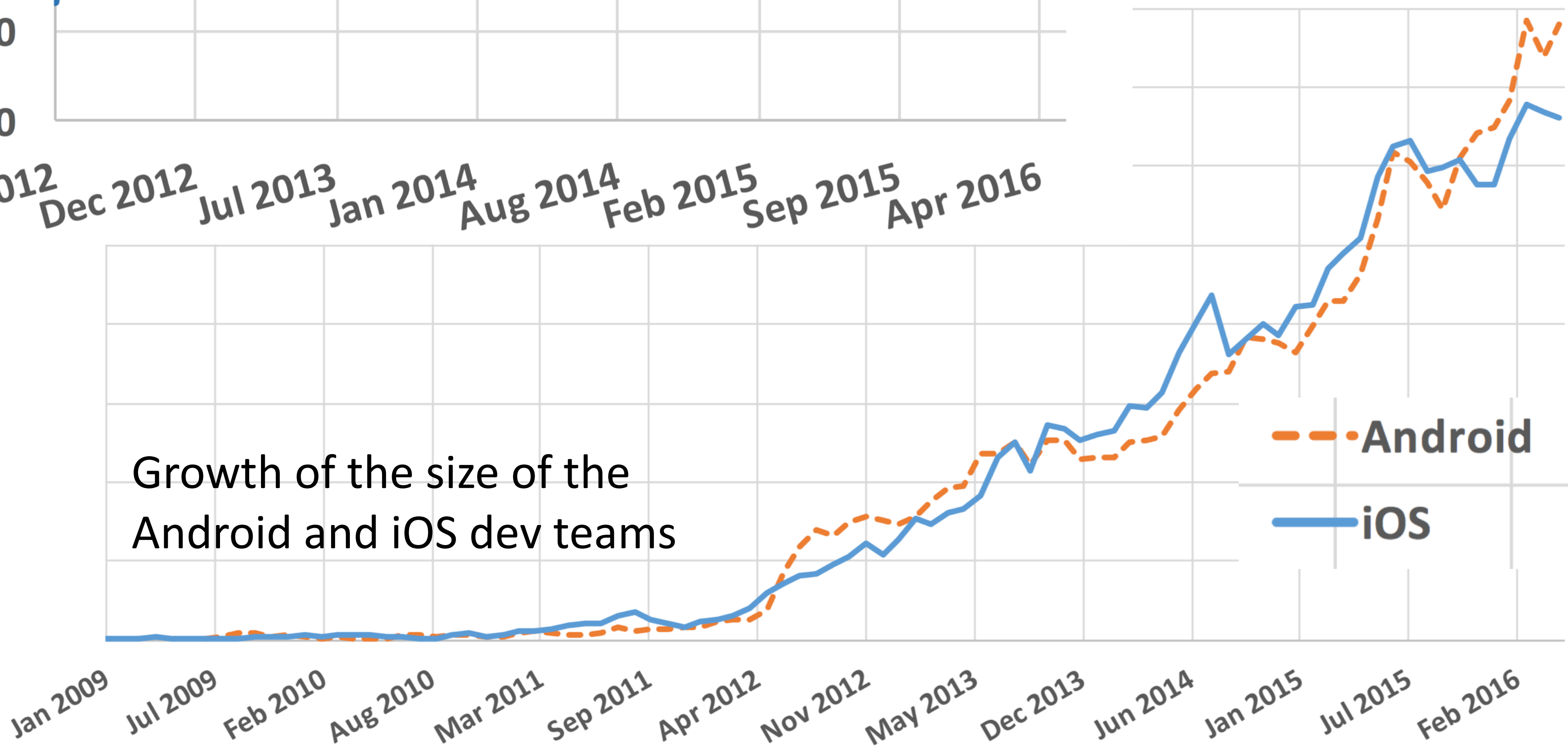
Combined with a centralized repository, this defines the monolithic model

- Piper users work at "head", a consistent view of the codebase
- All changes are made to the repository in a single, serial ordering
- There is no significant use of branching for development
- Release branches are cut from a specific revision of the repository



Did it work? Yes. Sustained productivity at Facebook





Growth of the size of the Android and iOS dev teams

Summary

- Configuration management
  - Treat infrastructure as code
  - Git is powerful
- Release management: versioning, branching, …
- Software development at scale requires a lot of infrastructure
  - Version control, build managers, testing, continuous integration, deployment, …
- It's hard to scale development
  - Move towards heavy automation (DevOps)
- Continuous deployment increasingly common
- Opportunities from quick release, testing in production, quick rollback